This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

Pushing Server Bandwidth Consumption to the Limit: Modeling and Analysis of Peer-assisted VoD

Ke Xu, *Senior Member, IEEE*, Haiyang Wang, *Member, IEEE*,
Jiangchuan Liu, *Senior Member, IEEE*, Song Lin, *Student Member, IEEE*, and Lei Xu, *Student Member, IEEE*

***Abstract*—Recent years have witnessed *Video-on-Demand* (VoD) as an efficient means for providing reliable streaming service for the Internet users. It is known that such peer-assisted VoD systems, as NetFlix and PPlive, generally incur a lower deployment cost in terms of server bandwidth consumption. However, some fundamental issues still need to be further clarified especially for VoD service providers. In particular, how far can we push peer-assisted VoD forward, and at the scale of VoD systems, the maximum reduction of server bandwidth consumption that can be achieved with peer-assisted approaches.**

In this paper, we provide an extensive model analysis to understand the minimum server bandwidth consumption for peer-assisted VoD systems. We first propose a basic model that can optimally schedule user demands at given snapshots. Our model analysis reveals the optimal performance bound and shows that the existing peer-assisted protocols are still far from being optimal. How to push the server bandwidth consumption to the limit remains a big challenge in the VoD system design. To approach the optimal bandwidth consumption in real deployment, we further extend our model to a realistic case to capture the peer dynamic across continuous time-slots. The simulation result indicates that the optimal load scheduling problem is still achievable through a dynamic programming algorithm. Its design principle further motivates a fast priority-based algorithm which achieves near-optimal performance. These proposed algorithms can significantly reduce the bandwidth consumption of dedicated VoD servers.

***Index Terms*—Video-on-demand (VOD), peer-assisted systems, scheduling.**

I. Introduction

In the past decade, peer-assisted *Video-on-Demand* (VoD) has become one of the most popular applications over the Internet. We have witnessed the successful deployments of such commercial systems as NetFlix[1] and PPlive[2]. Although these peer-assisted systems generally incur a lower deployment cost, the video publishers still need to deploy a great number of dedicated servers to ensure their service reliability. For example, Netflix has deployed more than $77,000$ video servers globally [1], and it is still planning to obtain more service capacity from the cloud platforms [2]. It is easy to see that the elevating user demands significantly increase the server bandwidth consumption even in the peer-assisted VoDs. Nowadays, the rising popularity of cloud-based applications also attracted studies to dynamically scale the service capacities [3]. Yet, leasing cloud resources will still introduce extra costs to the service providers. It is thus important to see how far can we push peer-assisted VoD forward. In particular, what is the lower bound of server bandwidth consumption under the existing peer-assisted VoD scenario.

In this paper, we provide an extensive model analysis to understand the minimum server bandwidth consumption in peer-assisted VoD systems. We first propose a basic model that can optimally schedule user demands at given snapshots. In particular, for a given amount of user demands, this model will schedule the load with minimum server bandwidth consumption. We transform the problem into a flow network and employ a modified maximum flow algorithm to obtain the optimal bandwidth consumption. To approach such an optimal bound in real deployment, we further extend our model to a more realistic case to capture the peer dynamic across continuous time-slots. We show that the optimal load scheduling problem is still solvable through a dynamic programming algorithm. The design of this approach further motivates a fast priority-based algorithm which achieves near-optimal performance. The simulation results indicate that the proposed algorithms can significantly reduce the bandwidth consumption of the dedicated VoD servers under different system configurations.

The rest of this paper is organized as follows: In Section II, we present the related work. Based on the basic model in Section 3, we investigate the optimal bandwidth consumption at given snapshots in Section 4 and try to approach such an optimal performance in Section 5 across continuous time-slots. Section 6 extensively evaluates the proposed algorithms and finally Section 7 concludes the paper.

II. Related Work

It is known that Huang *et al.* [4] for the first time discussed the design and potential benefits of peer-assisted VoD systems. After that, there have been numerous studies on the implementation, analysis, or optimization of the peer-assisted VoD applications [5], [6], [7].

Ke Xu is with the Department of Compute Science, Tsinghua University and Tsinghua National Laboratory for Information Science and Technology, Beijing, China.E-mail: xuke@tsinghua.edu.cn

Haiyang Wang is with the Department of Computer Science at the University of Minnesota Duluth, MN, US. E-mail: haiyang@d.umn.edu

Jiangchuan Liu is with the School of Computing Science, Simon Fraser University, British Columbia, Canada. E-mail: jcliu@cs.sfu.ca

Song Lin is with EatStreet Inc., Madison, WI, US, this work has been done when he was doctor candidate in Tsinghua University. E-mail: linsong1984@gmail.com

Lei Xu is with the Department of Computing Science, Tsinghua University, Beijing, China. E-mail: l-xu12@mails.tsinghua.edu.cn

[1]https://www.netflix.com/

[2]https://www.pptv.com/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

For VoD measurement and system design, Cheng *et al.* [6] presented a comprehensive study on the effectiveness and user experience of P2P-VoD systems. Based on the measurement, a large-scale P2P-VoD system is then presented in [5], which extends PPLive, one of the most successful peer-to-peer (P2P) live streaming systems. Another working system is GridCast [7], which is deployed on China Education and Research Network (CERNET). Feng *et al.* [8], provided an in-depth analytical understanding of fundamental properties of P2P streaming systems, with a particular spotlight on the benefits of network coding. There are also many studies focusing on the system optimization of VoD systems. For example, some consider the optimization of peer selection strategies [9], [10], [11] and others aim to optimize the server bandwidth consumption [12], [13], [14]. For example, Wu *et al.* [15] addressed some fundamental issues such as the optimal replication ratio in VoD systems. Ciullo *et al.* [16] provided an analytical methodology to design efficient peer-assisted VoD systems and optimal resource allocation strategies under server capacity constraints.

Zhou *et al.* [17] proposed a stochastic model that can be used to compare different download strategies. Liu *et al.* [12] analyzed how to distribute resources for the streaming systems to improve server capacity, video quality, and the depth of distribution trees that deliver the content. Niu *et al.* [3] developed a predictive resource auto-scaling system that dynamically books the minimum bandwidth resources from multiple data centers for the VoD provider to match its short-term demand projections. Tan *et al.* [18] investigated the problem of content placement in P2P systems, with the objective of maximizing the utilization of peer's uplink bandwidth resources. Suh *et al.* [19] investigated the design of a push-to-peer video-on-demand system in cable networks. Parvez *et al.* [20] developed a theoretical model to analyze the performance of BitTorrent-like protocols for on-demand stored media streaming. Liu *et al.* [21] constructed a fine-grained stochastic supply-demand model to investigate peer caching and prefetching as a global optimization problem. This study not only provides insights in understanding the fundamental demand characteristics, but also offers guidelines toward optimal prefetching and caching strategies in peer-assisted on-demand streaming systems.

In terms of server bandwidth costs, it is known that peer-assisted VoD, with the proper prefetching policy, can dramatically reduce server bandwidth costs [6]. In particular, Huang *et al.* [4] showed that peer-assistance can dramatically reduce server bandwidth costs, particularly if peers prefetch content when there is spare upload capacity in the system. Ciullo *et al.* [22] proposed a stochastic fluid framework that allows to characterize the additional bandwidth requested from the servers to satisfy all users watching a given video. For example, Pawel *et al.* [23] showed that for YouTube-like systems, the proposed optimizations would result in saving of as much as 70% of the server bandwidth [23]. Yu *et al.* [24] modeled the data scheduling problem in P2P systems and proposed an optimal scheduling scheme. Their approach is designed to maximize the playback continuity of all the users. Although this study also considered the server stress issues as one of their objectives, the lower bound of server bandwidth consumption remains largely unclear, not to mention the more complex case under the continuous time scenario. Different from studies such as [25] and [24], we for the first time estimate the optimal bound of server bandwidth in a continuous time analysis.

III. Problem Outline and Definition

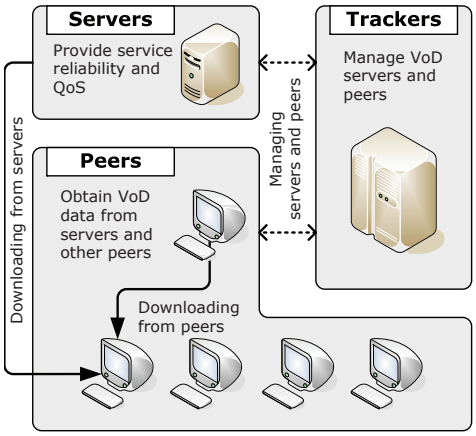

Fig. 1: Basic framework of peer-assisted VoD

It is known that the design of peer-assisted VoD can incur a lower deployment cost. However, some fundamental issues remain largely unclear for the VoD service providers. For example, the maximum reduction of server bandwidth consumption[3] that can be achieved with the peer-assisted approaches.

Figure 1 presents the basic framework of peer-assisted VoD systems, where the solid lines refer to the data flow and the dotted lines show the control flow. It is easy to see that the peers can obtain VoD data from both servers and other peers. In other words, if we can make the best use of the available resource from the peers, the bandwidth consumption of the VoD servers can be naturally minimized. Addressing such a problem is by no means trivial especially considering the peer dynamics in the VoD systems. Therefore, we will model the problem step by step. Our objective is to minimize the server bandwidth consumption.

To facilitate our discussion, we list the key notations in Table I. There are m chunks in a given VoD system (distributing a given video content). The set of m chunks is denoted as $C = \{c_1, c_2, \ldots, c_m\}$, and the playback rate of chunk c_i is $\mathrm{Rate}(c_i)$. There are also n peers, $P = \{p_1, p_2, \ldots, p_n\}$, in the system. The maximum upload capacity of peer p_j is $\mathrm{Upband}(p_j)$. We use T to denote the time-slot and use $R(t)$ to denote the chunk requirements at time t. In particular, $R(t_1)$ is a $m*n$ binary matrix. Each element in $R(t_1)$ denotes whether a given chunk is required by a given peer (1: yes, 0: no). We also use $O(t, c)$ to refer the set of peers that own the chuck c at time t. Moreover, with any $L \subseteq C$, we define

$$u(t, L) = \sum_{p \in \bigcup\limits_{c \in L} O(t,c)} \mathrm{Upband}(p)$$

[3]We assume that the bottleneck is at the edge of the networks. The disk I/O bandwidth is also not considered in our study.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

TABLE I: List of Notations

C	Set of chunks
$Rate$	Playback rate of a chunk
T	Set of time-slots
R	Chunk demand of peers
O	Set of peers that have certain chunks
$Prefetch$	Prefetching set
ϕ	Initial configuration of the prefetching set
u	Total upload capacity of peers
d	Total download capacity of peers
P	Set of peers in the system
k	Total number of cached chunks per peer
η	Total number of peers with download demands

$$d(t, L) = \sum_{r \in R(t) \wedge \text{Chuck}(r) \in L} \text{Rate}(\text{Chuck}(r))$$

where u refers to the total upload capacity of peers[5] and d refers to the total demands. $u(t, \phi) = d(t, \phi) = 0$.

Figure 2 shows an example with 3 peers and 3 chunks at time t where $C = \{c_1, c_2, c_3\}$, $P = \{p_1, p_2, p_3\}$, $O(t, c_3) = \{p_2, p_3\}$, $u(t, \{c_3\}) = \text{Upband}(p_2) + \text{Upband}(p_3)$.

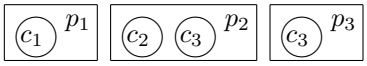

Fig. 2: An example for the notations

It is worth noting that we consider the problem at the chunk level in this model. Peers only upload chunks that have been completely downloaded [5]. They will also prefetch the contents when there is enough upload capacity in the system. We adopt the widely-used multiple video approach model. In this case, a peer could redistribute videos which either are being watched, or were watched previously. It is important to point out that many deployed P2P streaming systems are using the multiple video approach [26].

We assume that the servers have all the video chunks. The playback time for a chunk is set to be constant, which is the length of a time slice. We also assume that the server and the peers are able to upload to as many peers as needed, without constraints on the number of simultaneous connections. Moreover, the transmission bottleneck is at the edge of the networks [12].

IV. Minimizing Server Bandwidth Consumption: Optimal Bound at Given Snapshots

In this section, we will investigate the minimum server bandwidth consumption at given snapshots. This can help us to understand the maximum reduction of server bandwidth consumption with the peer-assisted approaches. Note that bandwidth capacities, cache states of peers and servers, and requirements of peers are known in this ideal case.

[5] $u(t, c)$ indicates the total upload capacity of chunk c at time t.

A. Utilizing Service Capacity from Peers

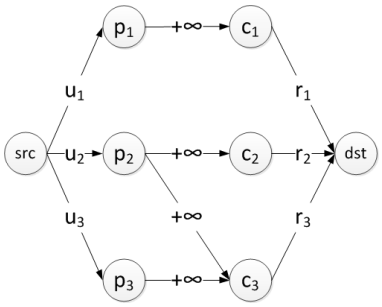

Fig. 3: Network flow

To make the best use of available resources on the peers, we convert the problem into a maximum flow problem in a flow network. As shown in Figure 3, the nodes bandwidth constraints are transferred into edge capacities. Without loss of generality, we also add a virtual source and a virtual sink in the flow network $G(V, E)$ where

$$V = \{\text{src}, \text{dst}\} \cup \{p_i | 1 \leq i \leq n\} \cup \{c_j | 1 \leq j \leq m\}$$

All the edges and their capacities are listed below:

$$\text{cap}(\text{src} \rightarrow p_i) = \text{Upband}(p_i)$$

$$\text{cap}(p_i \rightarrow c_j) = \infty, \text{if } p_i \in O(t, c_j)$$

(Note: if $p_i \notin O(t, c_j)$, there is no such edge.)

$$\text{cap}(c_j \rightarrow \text{dst}) = d(t, \{c_j\})$$

In this graph, when peer p has chunk c at the time-slot t, the edges between p and c will be configured to have infinite capacity. Otherwise, there will be no edge between p and c. This is because peer's upload and download capacities are already modeled by the source/desintation-related edges. Note that our objective is to find the maximum network flow for $\text{src} \rightarrow \text{dst}$. In particular, the corresponding graph for the example in Figure 2 is shown in Figure 3.

Note that the maximum bandwidth capacity that peers can provide equals to the maximum network flow for $\text{src} \rightarrow \text{dst}$, which can be denoted as $\text{MaxFlow}(\text{src} \rightarrow \text{dst})$. Since there is a one-to-one relationship between a peer transfer scheduling and a network flow in this graph, the flow amount from p_i to c_j is the bandwidth that peer p_i uploads to c_j while the capacities of $\text{src} \rightarrow p_i$ limit the upload bandwidth. The capacities of $c_j \rightarrow \text{dst}$ limit the download requirements. The maximum bandwidth that peers could provide equals to $\text{MaxFlow}(\text{src} \rightarrow \text{dst})$. Therefore, the minimum server bandwidth requirement is:

$$\S_{\min}(t) = d(t, C) - \text{MaxFlow}(\text{src} \rightarrow \text{dst}) \quad (1)$$

The best algorithms to solve the maximum flow problem include relabel-to-front [27] with a time complexity of $O(V^3)$ and the binary blocking flow algorithm [28] with a time complexity of $O(\min(V^{2/3}, E^{1/2}) E \log(V^2/E) \log U)$. Note

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

that the capacities of edges are integers in $[0, U]$. Therefore the problem can be solved with the worst running time of $O((m+n)^3)$. Based on Equation 1, we can also get that the minimum server bandwidth consumption is the sum of download requirements minus the maximum bandwidth that peers could provide. The related proof is as follows:

Theorem 1. *Denoting the minimum bandwidth that server should provide as $S_{\min}(t)$, we have*

$$S_{\min}(t) = \max_{L \subseteq C}(d(t, L) - u(t, L))$$

Proof: According to the max-flow min-cut theorem, the maximum network flow for $s \to t$ equals to the min-cut of the graph. For a cut $(\hat{S}, \hat{T})$, if there is $p \in \hat{S}$ and $m \in \hat{T}$ such that $p \in O(m)$, the capacity of the cut is infinite. So, it cannot be the min-cut. Considering all the other cuts, we let

$$L = \hat{T} \cap C$$

$$R = P \backslash \hat{S}$$

$$R^\star = \bigcup_{c \in L} O(t, c)$$

where L is a subset of C. We can see that there is no infinite path in the cut. We therefore have

$$R^\star \cap \hat{S} = \phi$$

$$R^\star \subseteq R$$

Therefore, the edges across $\hat{S}$ and $\hat{T}$ have the capacity sum

$$\sum_{c_i \in C \backslash L} d(t, c_i) + \sum_{p \in R} u(t, p) = d(t, C \backslash L) + \sum_{p \in R} u(t, p)$$

For a fixed L, it has the minimum value

$$d(t, C \backslash L) + u(t, L)$$

which can be achieved when

$$R = R^\star$$

i.e.,

$$\hat{T} = \{\mathrm{dst}\} \cup \mathrm{L} \cup \mathrm{R}^\star$$

If we traverse over all the subsets L of C, we have the capacity of the min-cut as follows:

$$\min_{L \subseteq C}(d(t, C \backslash L) + u(t, L))$$

The minimum server bandwidth is the sum of download requirements minus the maximum bandwidth that peers could provide

$$d(t, C) - \min_{L \subseteq C}(d(t, C \backslash L) + u(t, L)) = \max_{L \subseteq C}(d(t, L) - u(t, L))$$

So the theorem holds. ∎

In Figure 4, we present an example for cut corresponding to the example in Figure 2 where $\hat{S} = \{\mathrm{src}, p_3, c_3\}$, $\hat{T} = \{p_1, c_1, p_2, c_2, \mathrm{dst}\}$, $L = \{c_1, c_2\}$, $R = \{p_1, p_2\}$, $R^\star = \{p_1, p_2\}$. This is an example with 3 peers and 3 chunks at time t.

Note that Theorem 1 can be used to better understand how the parameters affect the system performance. However, it is hard to obtain the maximum amount of bandwidth that the peers can get from all the other peers.[6] To address this problem, we give an approximation as follows:

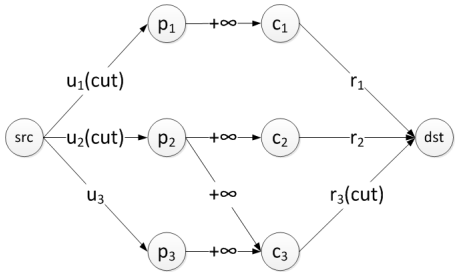

Fig. 4: An example for cut

Theorem 2. *When the download requirements are uniformly distributed among all the chunks, the maximum of the bandwidth that all peers can obtain (from all other peers) can be approximated as:*

$$\mathrm{Peer}_{\max}(t) \approx \min(\mu_r n p_r, n\mu_u(1 - e^{-\frac{knp_r}{m}}))$$

where μ_u is the expectation of peer upload capacity and μ_r is the expectation of download requirements. It is worth noting that we assumed a uniformly distributed peer demands among all the chunks. This is to better analysis the maximum bandwidth that all peers can obtain in an idea case. In the case of some real-world systems, their skewed user demands will decrease the total available bandwidth, and Theorem 2 can still serve as a useful upper-bound for optimization.

Proof: According to the definition of n and p_r, the number of peers which has download requirements is

$$|R(t)| \approx np_r$$

According to Theorem 1,

$$\mathrm{Peer}_{\max}(t) = \min_{L \subseteq R(t)}(d(t, R(t) \backslash L) + u(t, L))$$

where $|L| = h$ denotes the number of shaded chunks. We can therefore obtain $\mathrm{cc}(h, m)$ as follows. It indicates the expectation of taking h repeatable items (from m items).

$$\mathrm{cc}(h, m) = \sum_{j=1}^{\min(m,h)} \left(jC_m^j \sum_{i=1}^{j} (-1)^{j-i} C_j^i (\frac{i}{m})^h \right) \quad (2)$$

We find that

$$\mathrm{cc}(h, m) \approx m - me^{-\frac{h}{m-0.5}} \approx m(1 - e^{-\frac{h}{m}})$$

Further, for a given L, the probability that a peer contains at least one of the chunks in L is

$$1 - (1 - \frac{\mathrm{cc}(h, m)}{m})^k \approx 1 - e^{-\frac{kh}{m}}$$

$u(t, L)$ is the product of μ_u and the number of peers that contain at least one chunk in L, i.e.,

$$u(t, L) = n\mu_u(1 - e^{-\frac{kh}{m}})$$

[6]This is because when we consider all peers and the possible combinations between peers and chunks, whether a combination can generate the graph with the maximum amount of bandwidth cannot be validated unless we try all the combinations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

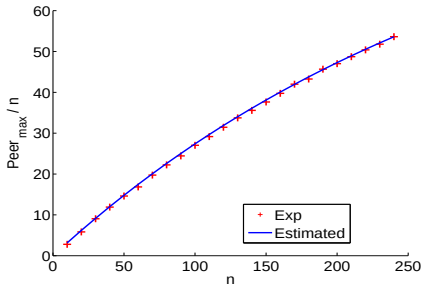

Fig. 5: $\frac{\text{Peer}_{\max}}{n}$ for different n

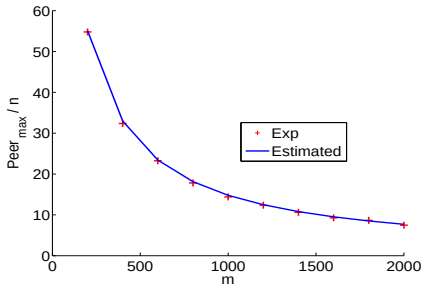

Fig. 6: $\frac{\text{Peer}_{\max}}{n}$ for different m

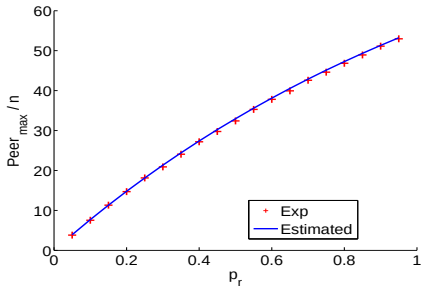

Fig. 7: $\frac{\text{Peer}_{\max}}{n}$ for different p_r

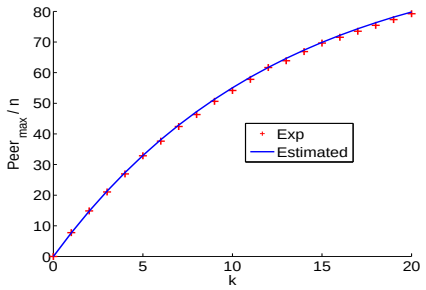

Fig. 8: $\frac{\text{Peer}_{\max}}{n}$ for different k

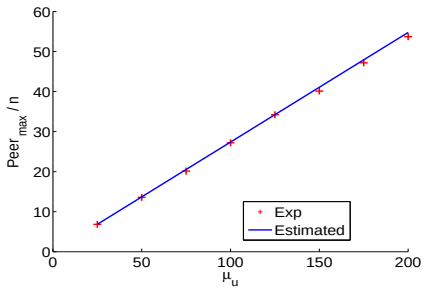

Fig. 9: $\frac{\text{Peer}_{\max}}{n}$ for different μ_u

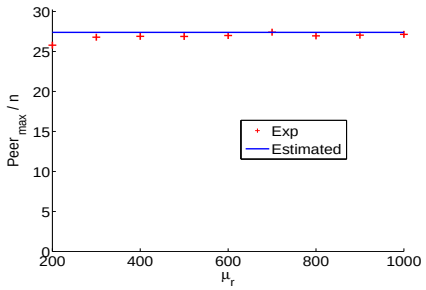

Fig. 10: $\frac{\text{Peer}_{\max}}{n}$ for different μ_r

$$d(t, R(t)\backslash L) = \mu_r(|R(t)| - h)$$

For a fixed h, $d(t, R(t)\backslash L)$ and $u(t, L)$ are similar among L with $|L| = h$, thus

$$\text{Peer}_{\max}(t) \approx \min_{0 \le h \le |R(t)|} (\mu_r(|R(t)| - h) + n\mu_u(1 - e^{-\frac{kh}{m}}))$$

The minimal value of the function value for integers in an interval can be approximated by the minimal value of the function value for all the real numbers in the interval, thus

$$\text{Peer}_{\max}(t) \approx \min_{0 \le x \le np_r} (\mu_r(np_r - x) + n\mu_u(1 - e^{-\frac{kx}{m}}))$$

$$\text{Peer}_{\max}(t) \approx \min(\mu_r np_r, n\mu_u(1 - e^{-\frac{knp_r}{m}}))$$

the theorem is proved. ■

Figures 5 to 10 give a comparison between our approximations and the accurate results obtained by the network flow approach. We apply the most typical values in a small VoD system [4] in this simulation:

$$m = 500, n = 100, p_r = 0.4, k = 4, \mu_u = 100, \mu_r = 600.$$

We run 1000 test cases for each parameter, and show their average as the result of the network flow method. It can be seen that the approximate results are very close to the accurate results. Moreover, the average download bandwidth that a peer can get (from other peers) is increasing in accordance with larger n, p_r, k and μ_u.

B. Analysis of Server Bandwidth Consumption

Based on the model analysis, we have clarified the maximum upload capacity of peers. In this subsection, we will further explore the optimal server bandwidth consumption and compare it with some widely-used peer-selection strategies: the random peer selection and the greedy peer selection. Note that the random algorithm is the key feature in PPLive's neighbor selection [26]. The details of the random algorithm and the greedy algorithm are shown in Table II and Table III respectively. In these algorithms, the peers upload chunks "as many as possible" means that peer's upload rates will be identical to their maximum upload capacities.

TABLE II: The Random Algorithm

```
The Random Algorithm
while there are chunks not yet satisfied
    Randomly select a chunk c* that not yet satisfied
    if there are peers that can satisfy c*
        Randomly select a peer p* to upload c* as many as possible
    else
        Let the server satisfy the remaining request of c*
    end if
end while
```

TABLE III: The Greedy Algorithm

```
The Greedy Algorithm
z(c_i) = Σ_{p_j ∈ O(c_i)} u(p_j)
while there are chunks not yet satisfied
    for every chunk c_i
        if there are peers that can satisfy c_i
            Send requests to all the peers that can upload c_i
        else
            Let the server satisfy the remaining request of c_i
        end if
    end for
    for every upload peer p(j)
        From all the requests to p(j), select c*
            with the minimal z(c)
        p(j) uploads c* as many as possible
    end for
end while
```

To obtain the optimal server bandwidth consumption, we need to obtain several parameters such as m, n, $\hat{d}$, u and O, where $\hat{d}_i$ is the total upload bandwidth required for a chunk c_i. In this paper, we make assumptions on the distributions of $\hat{d}$, u and O as follows. We assume $n\eta$ out of n peers need to download some chunks. Since

$$\hat{d}_i = \text{Rate}(c_i) \cdot v(c_i) \quad (3)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

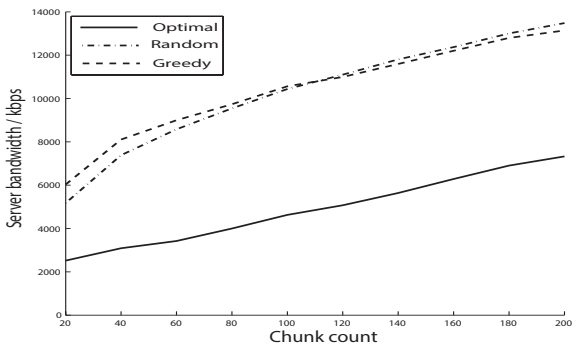

Fig. 11: Server bandwidth for different chunk counts

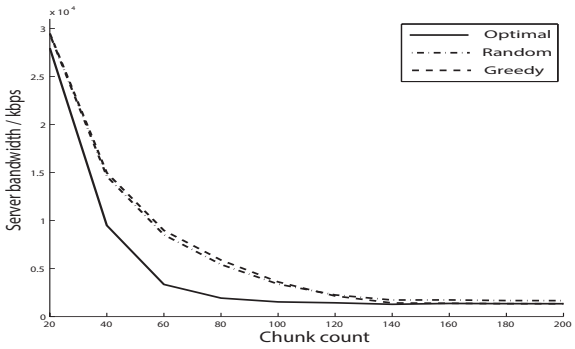

Fig. 12: Server bandwidth for different m and k

where $\mathrm{Rate}(c_i)$ is the playback rate of c_i, and $v(c_i)$ is the number of peers watching c_i. We assume that $\mathrm{Rate}(c_i)$ follows a normal distribution with mean μ_r and standard deviation σ_r i.i.d., and $v(c_i)$ follows a Zipf [29] distribution with parameter α and holds

$$\sum_{i=1}^{m} v(c_i) = n\eta \tag{4}$$

If $\eta < 1$, let $\mu_r' = \mu_r\eta$, $\sigma_r' = \sigma_r\eta$, $\eta' = 1$, we have $r' = r\eta$, $v' = v/\eta$ and $\hat{d}' = \hat{d}$. As the same test case can be generated, we assume $\eta = 1$ in the following analysis.

Assume that all $u(p_j)$ follow a normal distribution with mean μ_u and standard deviation σ_u, i.i.d. It is known that the total replications of a chunk c_i in the system should be approximately proportional to the download request $\hat{d}_i$. We can see that if some less popular chunks have more replications than the proportional value, the system will perform better. To explore such a problem, we assume that the total replication number of a chunk c_i in the system is proportional to $\hat{d}_i^q$ where q indicates the replication policy. Meanwhile, $q = 1$ indicates the proportional replication policy is applied in the system. Therefore, we can get the probability that p_j is an element of set $O(c_i)$ as follow:

$$\min\left(\frac{\hat{d}_i^q}{\sum_{i=1}^{m} \hat{d}_i^q} \cdot k, 1\right) \tag{5}$$

where k is the average number of chunks in the cache of a single peer.

Based on these definitions, we use Table IV as the input of our simulation. For every group of parameters, we run 500 test cases of $(m, n, \hat{d}, u, O)$. For each test case, we obtain the optimal server bandwidth consumption and compare it with the random and greedy algorithms.

TABLE IV: Parameters Used in Simulation

Parameter	Value
m	100
n	200
α	1
μ_r	400 kbps
σ_r	50 kbps
μ_u	400 kbps
σ_u	100 kbps
k	3
q	0.9

Figure 11 shows the server bandwidth consumption with different number of chunks. We can see that the performance of random and greedy algorithms are far from being optimal. In particular, these two algorithms will use almost twice of the optimal server bandwidth consumption. Moreover, a large number of chunks will decrease the probability that a peer owns a specified chunk; this will also increase the server bandwidth consumption.

As shown in Figure 12 ($k = m/20$), we have confirmed that if k increases proportionally to m, the optimal server bandwidth would remain the same for relatively large m and k. Note that this figure shows the greedy and random algorithms both have performance issues under certain configurations. For example, the gap between random and optimal will increase with smaller k/m ratios. This means only a small fraction of chunks can be cached. We can hardly obtain good performance unless we can cache all chunks ($k = m$).

Figure 13 presents the server bandwidth consumption with different number of peers. We can again find a large gap between the optimal benchmark and the random/greedy algorithms. It is worth noting that if we carefully optimize the relationship between peers, the total number of peers does not significantly increase the optimal server bandwidth in peer-assisted VoD systems.

Figure 14 shows the server bandwidth consumption with a different Zipf parameter α. A small α indicates that the requests to different chunks are the same, while large α indicates that requests mainly concentrate on several chunks. We can see that the performance gap will become extremely large with smaller α. Note that the existing measurement studies already showed that the video popularity matches the Zipf distribution [30].

Figure 15 presents the server bandwidth consumption with the average playback rates μ_r. We can see that for the random/greedy algorithms, the server bandwidth consumption increases significantly when the playback rates become larger than 300 $kbps$. The optimal benchmark, on the other hand, increases when the playback rates become larger than 400 $kbps$.

Figure 16 further presents the server bandwidth consumption with skewed distributed playback rates where the random/greedy algorithms use almost twice of the optimal server bandwidth consumption.

Figure 17 shows the server bandwidth consumption with different k, the average of cached chunks per peer. We observe that the greedy algorithm and the random algorithm cannot make full use of the upload capacity of peers.

We can also find that if each peer has 6 out of 100 chunks,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

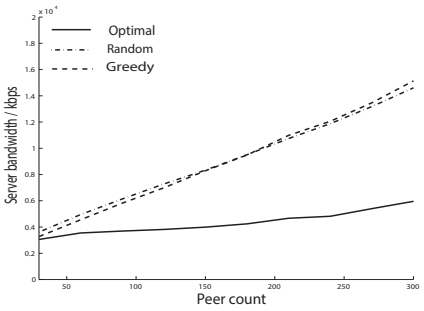

Fig. 13: Server bandwidth for different peer counts

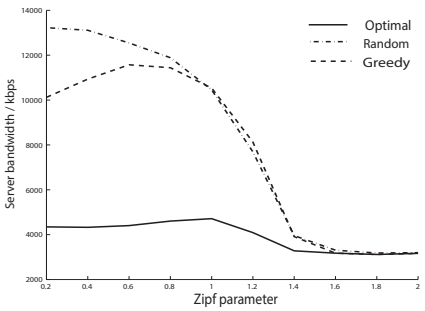

Fig. 14: Server bandwidth for different Zipf parameters

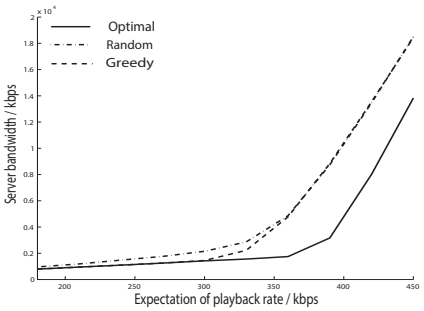

Fig. 15: Server bandwidth for different expectations of playback rates

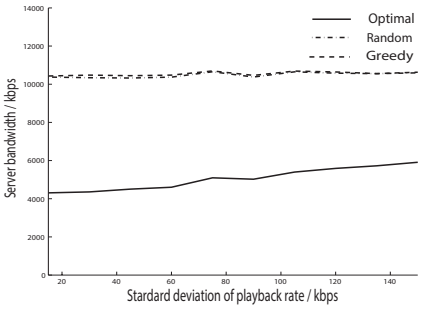

Fig. 16: Server bandwidth for different standard deviations of playback rates

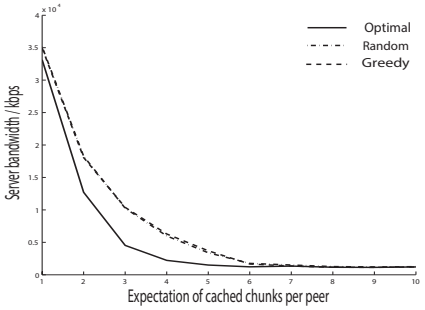

Fig. 17: Server bandwidth for different k

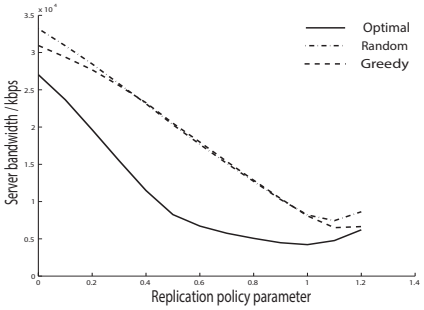

Fig. 18: Server bandwidth for different q

the server bandwidth can be kept at a modest level even with the simplest peer selection algorithm.

In Figure 18, we further discussed the server bandwidth consumption across different q. We can see that the minimal server bandwidth consumption is achieved when q is around 0.5.

We discuss static environment above, in the following sections we relax this condition.

V. Obtaining Optimal Bandwidth Consumption across Continuous Time-Slot

In the previous section, we have discussed the optimal bound of VoD server bandwidth consumption. It is easy to see that the existing peer-assisted protocols are still far from being optimal. Therefore, whether the optimal bandwidth consumption can be achieved in real-world deployment remains largely unknown across continuous time-slots. To calculate the minimum server bandwidth in the continuous time-slot scenario, an intuitive approach is to make sure that the bandwidth requirements can be met (no less than $S_{\min}$) over infinite time-slots. This indicates that we can perform a binary search on real numbers to find the result with a predetermined inaccuracy. In this section, we will first discuss the lower bound as well as the upper bound of $S_{\min}$. After that, we will carry out a search across infinite time-slots to find the optimal result.

A. Binary Search Framework

As shown in Table V, the key to accomplish this binary search is to design an algorithm to solve a feasibility problem: whether all the requirements can be met when the server has a bandwidth of w in every time-slot. If w is enough, $S_{\min}$ is no greater than w; we can therefore set the new upper bound to be w. Otherwise, $S_{\min}$ is greater than w, and we can set the new lower bound to be w.

TABLE V: Binary Search

Binary Search Framework
initialize upper and lower
while $\text{upper} - \text{lower} \geq \text{PredeterminedInaccuracy} * 2$ **do**
$\text{mid} = (\text{upper} + \text{lower})/2$
if checkContTime(mid)
$\text{upper} = \text{mid}$
else
$\text{lower} = \text{mid}$
end while
return $\frac{\text{upper}+\text{lower}}{2} \pm \frac{\text{upper}-\text{lower}}{2}$

The initial lower bound and upper bound do not affect the performance of our algorithm remarkably. The initial lower bound can be set as zero or the bandwidth required in the first time-slot. Since there is no prefetch before this time-slot, the bandwidth required can be calculated by the snapshot algorithm. The initial upper bound is the bandwidth required in any scheduling. For example, we can use the no-prefetch scheduling provided in the previous section (Section IV). We can also use the priority-based scheduling provided in Subsection V-D.

B. Feasibility Problem

The main process to this feasibility problem is provided in Table VI. In order to avoid reduplication calculation, dynamic programming is used. We consider the problem reversely, from the last time-slot to the first time-slot. We denote the power set of R as 2^R. In every time-slot, we need to know by prefetching which subsets of R in the previous time slots, the requirements would be satisfied. These subsets will form

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

a subset of 2^R, referred as "prefetching set" and denoted by $\mathrm{Prefetch}(t)$. With the initial value $\mathrm{Prefetch}(T) = \{\phi\}$, we can calculate $\mathrm{Prefetch}(t)$ using $\mathrm{Prefetch}(t+1)$, $t = T-1, T-2, \ldots, 1$. If by prefetching $R_1 \in R$, we can achieve some $R_2 \in \mathrm{Prefetch}(t+1)$, then R_1 can be put in $\mathrm{Prefetch}(t)$. Finally, since we cannot prefetch before the first time-slot, we only need to check whether $\phi \in \mathrm{Prefetch}(0)$ using $\mathrm{Prefetch}(1)$.

TABLE VI: Solution of Continuous Time Feasibility Problem

```
checkContTime function
function checkContTime(w)
Prefetch(T) = {φ}
foreach t in (T − 1), (T − 2), . . . , 1 do
  Prefetch(t) = calcNextPrefetch(w, Prefetch(t+1), t)
  if Prefetch(t) is empty
    return false
return checkOneSet(φ, w, Prefetch(1), 0);
```

Here we provide two approaches to optimize this framework. Firstly, note that, for any $R_3 \subset R_4 \subset R$, if we can prefetch R_4 before t using a prefetching set $R_5 \in \mathrm{Prefetch}(t-1)$, then we can also prefetch R_3 using R_5 before t. Removing R_4 from $\mathrm{Prefetch}(t)$ will not affect $\mathrm{Prefetch}(t-1)$ or the result. So by gradually removing supersets of other sets in $\mathrm{Prefetch}(t)$, we can finally get a $\mathrm{Prefetch}(t)$ where no set is a superset of another set, while not changing the result of the problem. This is very helpful in reducing calculation costs. As a special example, the best case is the time-slot can be satisfied without prefetching, since the empty set is a subset of any other prefetching sets. We do not need to check other sets, and $\mathrm{Prefetch}(t) = \{\phi\}$.

Secondly, to calculate $\mathrm{Prefetch}(t)$ using $\mathrm{Prefetch}(t+1)$, denote

$$\mathrm{Related}(t) = R(t) \cup \mathrm{Prefetch}(t+1)$$

When $\mathrm{Related}(t) \neq \phi$, a trivial prefetching is to prefetch every chunk in $\mathrm{Related}(t)$. If after the first optimization, we still have $\mathrm{Related}(t) \in \mathrm{Prefetch}(t)$, we cannot make any progress in the current time-slot and the problem result will be negative. Thus we do not put this trivial prefetching into $\mathrm{Prefetch}(t)$ when $\mathrm{Related}(t) \neq \phi$. Under this constraint, if during the process, $\mathrm{Prefetch}(t) = \phi$, which means it is impossible to satisfy the requirements after time t unless we prefetch everything, we can judge that the problem result is negative and do not need to check any previous time-slots. When $\mathrm{Related}(t) = \phi$, we have $\mathrm{Prefetch}(t) = \{\phi\}$ and still need to continue the calculation for previous time-slots.

These optimizations have modified the definition of "prefetching set". $\mathrm{Prefetch}(t)$ is "minimal", which means no set in $\mathrm{Prefetch}(t)$ is a superset of another set. When $\mathrm{Related}(t) \neq \phi$, $\mathrm{Related}(t) \notin \mathrm{Prefetch}(t)$. The first optimization remarkably reduces the size of "prefetching set" when server bandwidth is abundant, while the second optimization avoids unnecessary check when server bandwidth is not enough. Therefore in most cases the feasibility problem can be addressed very quickly. In Subsection V-C, we present further optimization to calculate the prefetching set under such modified definition.

C. Prefetching Set Calculation

To calculate $\mathrm{Prefetch}(t)$ using $\mathrm{Prefetch}(t+1)$, we have $\mathrm{Prefetch}(t) \subseteq 2^{\mathrm{Related}(t)}$. The naive solution is to check whether an element in $2^{\mathrm{Related}(t)}$ can be put in $\mathrm{Prefetch}(t)$, i.e., whether it can match any element in $\mathrm{Prefetch}(t+1)$. To check whether $R_1 \in \mathrm{Prefetch}(t)$, we can traverse over $\mathrm{Prefetch}(t+1)$. Using the snapshot result, if we figure out that we can achieve some $R_2 \in \mathrm{Prefetch}(t+1)$ by prefetching $R_1 \in \mathrm{Related}(t)$, then $R_1 \in \mathrm{Prefetch}(t)$. If we cannot achieve any R_2, then $R_1 \notin \mathrm{Prefetch}(t)$.

We can remarkably reduce the calculation cost in practice in two ways.

The first optimization is the order to traverse over possible subsets of $\mathrm{Related}(t)$, utilizing that $\mathrm{Prefetch}(t)$ is "minimal". First, we check whether $\phi \in \mathrm{Prefetch}(t)$. If so, it is done. Otherwise, we check every single element set in $\mathrm{Related}(t)$, and add those with positive results to $\mathrm{Prefetch}(t)$. We denote the union of single element set with negative results as $\mathrm{NotSolved}$. The left is to traverse over $2^{\mathrm{NotSolved}}$, ordered by the number of elements, and also add those with positive results to $\mathrm{Prefetch}(t)$. Before actually check a subset, we first check whether it is a superset of found $\mathrm{Prefetch}(t)$. The order of traversal guarantees that it will not be a superset of any not-yet-checked sets.

The second optimization is, before checking the match between $R_1 \in \mathrm{Related}(t)$ and $R_2 \in \mathrm{Prefetch}(t+1)$, when $R_1 \not\subseteq R_2 \cup R(t)$, which means if some prefetching is neither helpful in meeting the requirements in the current time-slot, nor helpful in prefetching contents for the time-slots later, we do not need to check this R_2. Since if the result is positive, it will be positive with a smaller prefetching set in $\mathrm{Prefetch}(t)$. This is not trivial after we have fixed R_1, and from the simulation we figure out that this optimization is also helpful in calculation efficiency.

These algorithms and optimizations are shown in Table VII.

TABLE VII: The Prefetching Set Calculation

```
calcNextPrefetch function
function calcNextPrefetch(w, Prefetch(t+1), t)
Traverse over R1 ∈ Related(t) in the optimized order
  Traverse over R2 ∈ Prefetch(t + 1)
  if R1 ⊈ R2 ∪ R(t)
    do not need to check this R2, continue
  if R1 matches R2
    R1 ∈ Prefetch(t)
    do not need to check the remaining R2s, break
```

D. Priority-based Scheduling

Based on the optimal solution derived in the single snapshot problem, we have shown that the optimal load scheduling problem is also solvable through a dynamic programming algorithm across continuous time. However, this algorithm cannot well satisfy the latency requirements in the real deployment. In particular, the dynamic-programming-based scheduling may cost more time than the chunk delivery of the VoD systems. To achieve higher efficiency in the real-world systems, we propose the following priority-based algorithm to further scheduling the chunk transmissions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

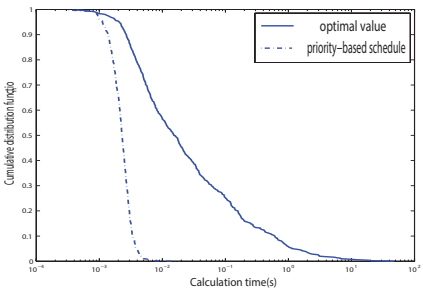

Fig. 19: CDF of the calculation time

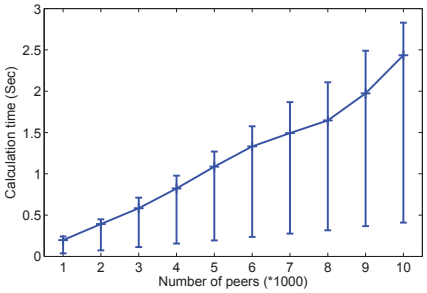

Fig. 20: Overheads of priority-based algorithm

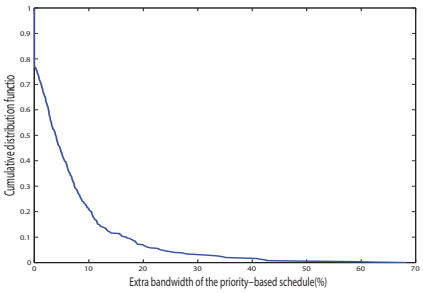

Fig. 21: Extra bandwidth of the priority-based scheduling (CDF)

It is known that the scheduling in each time-slot t should at least satisfy $R(t)$ in our model. The key problem is which requirements in the future time-slots we are going to satisfy. In the priority-based scheduling, we give each future requirement a priority. By strictly following the priority-based order, we try to satisfy as many requirements as possible. Since we can check whether a given set of requirements can be satisfied by the snapshot algorithm, we only need to do a binary search to find out how many requirements can be actually satisfied.

To set a proper priority, we notice that there are two benefits to satisfy the earlier requirement: firstly, it reduces the risk in the time-slots when we cannot satisfy all the requirements; secondly, after the time-slot the transfer is done, the cache status is broadcasted and has a longer time to serve more potential requirements in the future with a P2P transfer. Thus we order the future requirements according to their time-slots. For the requirements in the same time-slot, we will put them in a fixed order.

To calculate the bandwidth of the priority-based scheduling, binary search is again used with another feasibility problem: whether all the requirements can be met when the server has a bandwidth of w in every time-slot with this specified scheduling. This feasibility problem can be naively solved by simulation.

VI. Evaluation of Continuous-Time-Slot Approaches

In this section, we will evaluate the performance of our continuous-time-slot approaches. We will clarify the server bandwidth consumption under different schedulings. Their time complexities will also be investigated by comparing the worst-case running time.

A. Performance of Priority-based Scheduling

This subsection compares the optimal server bandwidth and the server bandwidth under the priority-based scheduling. It also compares the time to calculate these two results.

Since the simulation has involved efficiency issue, the computation environments are provided: the simulation is done on a laptop with 2.27GHz $\times$ 2 Intel CPU, 2.0 GB memory and Microsoft Win7 OS; the program language for the simulator is Microsoft C#.

In this simulation, we assume there are m movies, each with m_c chunks. If a peer is not requesting anything in a time-slot, it has a p_r probability to start a series of requests. The requests are targeted on all movies with equal possibility, with a length uniformly distributed among $1, 2, \ldots, m_c$.

For 500 test cases with $m_c = 6$, $m/m_c = 6$, $n = 6$ and $T = 5$, the average time to find the optimal server bandwidth within a 10^{-5} relatively inaccuracy is 0.437 s. The maximum calculation time is 49.4 s. The cumulative distribution functions (CDFs) of the time to find the optimal server bandwidth and priority-based scheduling results are shown in Figure 19. We can see that the priority-based scheduling can be calculated very fast. Figure 20 further investigates the case when there are more peers in the system. We can see that the time consumption is proportional to the number of peers.

On the other hand, on average, the priority-based scheduling is only 6.64% higher than the optimal value, while the worst case is 68% higher. In 23.2% of the cases, the priority-based scheduling actually achieves the optimal value. The extra bandwidth of the priority-based scheduling is shown in Figure 21. We can see that the extra bandwidth remains quite low in most cases.

B. Performance Comparison among Different Scheduling Approaches

In this subsection, we compare the server bandwidth under priority-based scheduling, the server bandwidth under no-prefetching scheduling, and the server bandwidth if we do not use P2P.

The results are given in Figures 22 to 27. Except for the parameter specified in each of the results, we use the typical values of a small VoD system for the parameters in the following:

$$m = 30, n = 20, T = 10, p_r = 0.5, \mu_u = 200,$$

$$\sigma_u = 50, \mu_r = 400, \sigma_r = 50.$$

For each group of parameters, we randomly generate 10,000 test cases. The server bandwidth of the three schedules are calculated for each test case, and the average results of 10,000 test cases are shown. We assume there is no replication in the initial state, and there is no cache replacement in the T time-slots. This assumption avoids the effects from the randomness in these factors.

Figure 22 gives the result for different chunk count m. If we only use servers, the bandwidth required is the sum of requirements, and thus not related to m. When m grows and the other variables remain unchanged, the probability of peer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

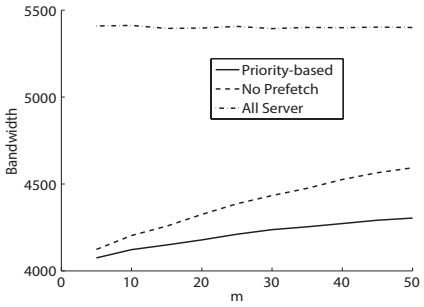

Fig. 22: Server BW for different m

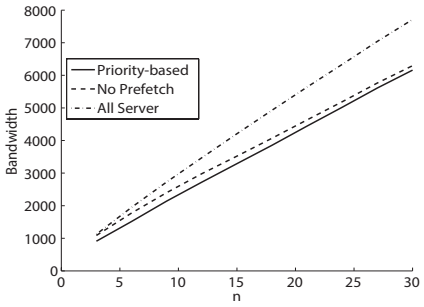

Fig. 23: Server BW for different n

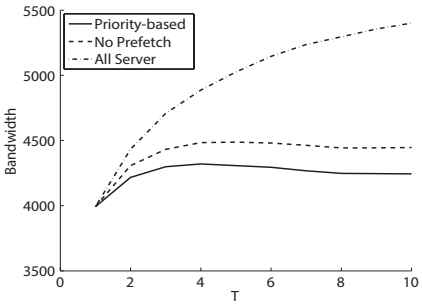

Fig. 24: Server BW for different T

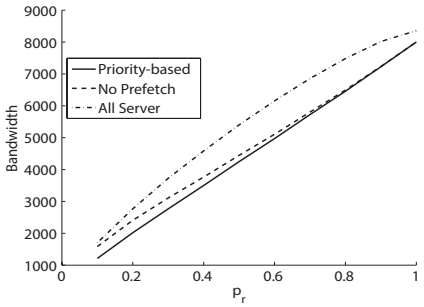

Fig. 25: Server BW for different p_r

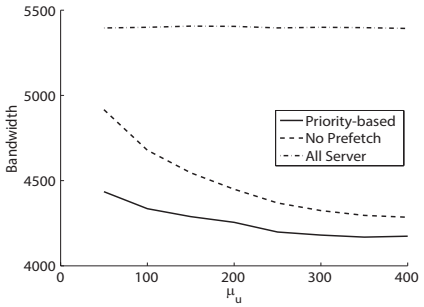

Fig. 26: Server BW for different μ_u

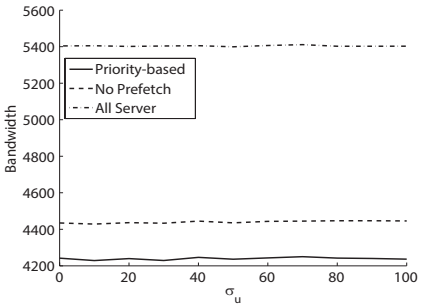

Fig. 27: Server BW for different σ_u

sharing will gradually decrease. So the bandwidth required will increase in the cases that P2P is used. But we can see that bandwidth growth with prefecting is slower than those without prefetching.

Figure 23 gives the result for different peer count n. The bandwidth in three cases are linear with n. By verifying the result, we notice that the growth speed of server bandwidth against n in priority-based scheduling and no-prefetch scheduling is similar, while the growth speed without using P2P is a bit faster.

Figure 24 gives the result for different time slot count T. As the peer has no initial cache, the results for one time slot should be the same. In these problems, the server bandwidth without P2P is the maximum of T i.i.d. variables, so it grows very fast with the increase of T. For the other two cases, when t is larger, there will be more replications among peers; the server bandwidth required in that time-slot may decrease. So in general, when T is larger, the server bandwidth in these two cases will remain at some value, or even decrease as T grows.

Figure 25 gives the result for different p_r. As p_r grows, the number of download requirements in the system is increasing, and the bandwidth for all the three cases is all increasing. It is worth noting that when the number of requirements grows to the situation that all the peer bandwidths are almost utilized, no-prefetching server bandwidth is close to the server bandwidth with prefetching.

Figure 26 gives the result for different μ_u. As μ_u grows, the bandwidth provided by peers will increase, so the server bandwidth for the cases with P2P are gradually decreasing. But even if peer bandwidth is relatively abundant, they may not be fully utilized in the no-prefetching case, where the server bandwidth required by all the peers is larger than the result for priority-based schedule.

Figure 27 gives the result for different σ_u. The results are little affected by σ_u from the perspective of the current parameters. For the above test cases, the server bandwidth under no-prefetching schedule has saved 15.3% server bandwidth, and the server bandwidth under priority-based schedule has saved 19.7% server bandwidth. But generally the test cases here are those that the requirements cannot be satisfied only by using peers. If the requirements are reduced, there will be cases in which P2P has saved the server bandwidth. Note that comparing to the snapshot model, we applied different parameters in the evaluation of continues time model. This is to show that our proposed method can achieve reasonable performance under different system configurations.

C. Performance in Planet-lab Deployments

To understand the performance of the priority-based schedule algorithm in real-world deployments, we further extend our evaluation to a Planet-lab [31] environment. In particular, we implement a VoD prototype system on Planet-lab with one server and 400 peers. These peers are initiated to generate segment request lists from random starting points. This list consists of video segments with the size of 4 Megabytes. After that, the peers will seek available bandwidth from other peers. If the demands cannot be fully satisfied by other peers, it will start to fetch the remaining segments from our VoD server. In this experiment, the basic configuration is identical to our previous simulation.[7]

Figure 28 shows the available server bandwidth when there are only 200 peers in the system. In this case, the maximum bandwidth on the VoD server is limited to 32 Mbps. As we can see from the figure, the available server bandwidth increases from 5 Mbps to 32 Mbps after 10 minutes. Figure 29 further investigates the the available server bandwidth when there are

[7]Our system prototype can also be found at: https://github.com/thuxl/vod-planetlab.git

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

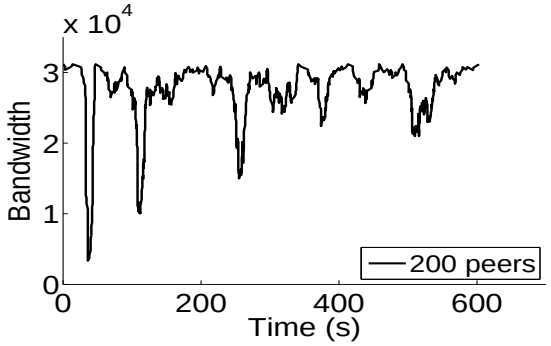

Fig. 28: Available server bandwidth with 200 peers

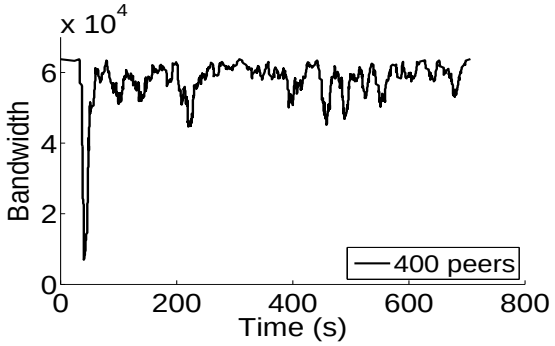

Fig. 29: Available server bandwidth with 400 peers

400 peers in the system. In this case, the maximum bandwidth is configured to 64 Mbps. It is easy to see that the available server bandwidth increase from 10 Mbps to 64 Mbps after 12 minutes. These real-world deployments indicate that the proposed priority-based schedule algorithm can successfully minimize the server bandwidth consumption with reasonable convergence time.

VII. Further Discussions

Our study has provided that, with careful optimization, the users can be further organized in a better way to achieve even lower bandwidth consumption on dedicated VoD servers. There are still many open issues for further exploration, and we list three that we are particularly interested in and consider to deploy into our model.

User Experience: This study provides a model-based analysis to clarify the server-side bandwidth consumption. To optimize the overall system performance, user experience, such as latency and video quality, also need to be further considered. This also includes user's local configurations (e.g., the value of k). Additionally, the changing requirements of users should be taken into consideration.

One of our ongoing measurements is to learn the user-side preferences via trace-analysis. We will apply real-world traces and dynamic network environment to better facilitate our future models analysis.

Social Relationship and Video Propagation Among VoD Users: The modern Social Networking Services (SNS) have drastically changed the information distribution landscape and people's daily life. With the development in broadband accesses, short videos as well as VoD videos have become one of the most important types of objects for social networking service users. However, how to obtain the social relationship and video propagation remains a big challenge among VoD users.

We are currently working on the epidemic models to accommodate the diversity of the VoD propagation. We believe that it will provide useful hints to design a more efficient VoD framework with better organization among peers.

Hybrid VoD with P2P and Cloud Deployment: Cloud computing has recently attracted a substantial amount of attentions from both industry and academia. This brings great opportunities for the VoD systems because the server capacities can now be dynamically provisioned based on user demands. Meanwhile, it also raises many design challenges. For example, how to design a generic framework that facilitates a cost-effective VOD migration to the cloud; and how to adjust cloud servers in a fine granularity to accommodate temporal and spatial dynamics of user demands.

Our ongoing work includes exploring the optimal solutions to deal with cloud servers with diverse capacities and lease prices, as well as the potential latencies in initiating and terminating leases in real world cloud platforms, etc.

VIII. Conclusion

In this paper, we explored the minimum server bandwidth consumption in a peer-assisted VoD system. We first proposed a basic model to optimally schedule the user demands at given snapshots. To approach the optimal bandwidth consumption in real deployment, we further extended our model to a more realistic case to capture the peer dynamics across continuous time slots. We showed that the optimal load scheduling problem is still solvable through a dynamic programming algorithm and further designed a faster priority-based algorithm to achieve a near-optimal performance. Our simulation results indicate that the proposed algorithms can significantly reduce the bandwidth consumption on the dedicated VoD servers. Note that the proposed priority-based algorithm is designed to optimize the general peer-assisted VoD system. Our next step is to implement the priority-based algorithm in such real-world frameworks as CoolStreaming [32] and further compare its efficiency with the prefetching protocols in these VoD systems.

Acknowledgment

This work has been supported in part by NSFC Project (61170292, 61472212), National Science and Technology Major Project (2012ZX03005001), 973 Project of China (2012CB315803), 863 Project of China (2013AA013302) and EU MARIE CURIE ACTIONS EVANS (PIRSES-GA-2013-610524).

References

[1] Netflix will ruin the internet! [Online]. Available: http://tech.fortune.cnn.com/2010/11/04/netflix-will-ruin-the-internet/

[2] Netflix gets its head in the cloudsand it's a great thing. [Online]. Available: http://www.activevideo.com/blog/2010/12/02/netflix-gets-its-head-in-the-clouds-and-its-a-great-thing/

[3] D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-assured cloud bandwidth auto-scaling for video-on-demand applications," in *Proceedings of IEEE INFOCOM*, 2012.

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2014.2360772, IEEE Transactions on Network and Science Management

[5] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proceedings of ACM SIGCOMM*, 2008.

[6] C. Huang, J. Li, and K. W. Ross, "Peer-assisted vod: Making internet video distribution cheap," in *Proceedings of the 6th International Workshop on Peer-to-Peer Systems*, 2007.

[7] B. Chen, L. Stein, H. Jin, and Z. Zhang, "Towards cinematic internet video-on-demand," in *Proceedings of EuroSys*, 2008.

[8] C. Feng and B. Li, "On large-scale peer-to-peer streaming systems with network coding," in *Proceedings of ACM Multimedia*, 2008.

[9] Y. Zhou, D. Chiu, and J. Lui, "A simple model for analyzing p2p streaming protocols," in *Proceedings of IEEE ICNP*, 2007.

[10] C. Wu and B. Li, "Optimal peer selection for minimum-delay peer-to-peer streaming with rateless codes," in *ACM P2PMMS*, 2005.

[11] Z. Ma, K. Xu, J. Liu, and H. Wang, "Measurement, modeling and enhancement of bittorrent-based vod system," in *Computer Networks, Elsevier, Special Issue on Measurement-based, Optimization of P2P Networking and Applications,*, vol. 56, no. 3, 2012, pp. 1103–1117.

[12] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *Proceedings of ACM SIGMETRICS*, 2007.

[13] Y. Tu, H. Zhong, M. Hefeeda, and S. Prabhakar, "An analytical study of peer-to-peer media streaming systems," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 4, pp. 354–376, 2005.

[14] H. Hlavacs and S. Buchinger, "Hierarchical video patching with optimal server bandwidth," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 1, pp. 1–23, 2008.

[15] W. Wu and J. Lui, "Exploring the optimal replication strategy in p2p-vod systems: Characterization and evaluation," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, 2012, pp. 1492–150.

[16] D. Ciullo, V. Martina, M. Garetto, E. Leonardi, and G. L. Torrisi, "Performance analysis of non-stationary peer-assisted vod systems," in *Proceedings of IEEE INFOCOM*, 2012.

[17] Y. Zhou, D. Chiu, and J. Lui, "A simple model for chunk-scheduling strategies in p2p streaming," in *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, 2011, pp. 42–54.

[18] B. Tan and L. Massouli, "Optimal content placement for peer-to-peer video-on-demand systems," in *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, 2013, pp. 566–579.

[19] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Peer-assisted on-demand streaming: Characterizing demands and optimizing supplies," in *IEEE J. Selected Areas in Comm.*, vol. 25, no. 9, 2007, pp. 1706–1716.

[20] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of bittorrent-like protocols for on-demand stored media streaming," in *Proceedings of ACM SIGMETRICS*, 2008.

[21] F. Liu, B. Li, B. Li, and H. Jin, "Peer-assisted on-demand streaming: Characterizing demands and optimizing supplies," in *IEEE Transactions on Computers*, vol. 62, no. 2, 2013, pp. 351 – 361.

[22] D. Ciullo, V. Martina, M. Garetto, E. Leonardi, and G.L.Torrisi, "Stochastic analysis of self-sustainability in peer-assisted vod systems," in *Proceedings of IEEE INFOCOM*, 2012.

[23] P. Garbacki, D. Epema, and J. Pouwelse, "Offloading servers with collaborative video on demand," in *Proceedings of the 7th International Workshop on Peer-to-Peer Systems*, 2008.

[24] Q. Yu, B. Ye, S. Lu, and D. Chen, "Optimal data scheduling for p2p video-on-demand streaming systems," in *IET Communications*, vol. 12, no. 6, 2012, pp. 1625–1631.

[25] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "On the optimal scheduling for media streaming in data-driven overlay networks," in *Proceedings of IEEE GLOBECOM*, 2006.

[26] K. Xu, H. Li, J. Liu, W. Zhu, and W. Wang, "PPVA: A universal and transparent peer-to-peer accelerator for interactive online video sharing," in *Proceedings of the 18th International Workshop on Quality of Service*, 2010.

[27] A. Goldberg and R. Tarjan, "A new approach to the maximum flow problem," in *Proceedings of the 18th annual ACM Symposium on Theory of Computing*, 1986.

[28] A. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *J. ACM*, vol. 45, no. 5, pp. 783–797, Sep. 1998.

[29] Z. George, *Human Behavior and the Principle of Least Effort*. Boston, MA, USA: Addison-Wesley, 1949.

[30] W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *In Proc. of ACM EuroSys*, 2006.

[31] Planet-Lab. [Online]. Available: https://www.planet-lab.org/

[32] CoolStreaming. [Online]. Available: http://www.coolstreaming.us/

Ke Xu (M'02-SM'09) received his Ph.D. from the Department of Computer Science and Technology of Tsinghua University, Beijing, China, where he serves as full professor. He has published more than 100 technical papers and holds 20 patents in the research areas of next generation Internet, P2P systems, Internet of Things (IoT), network virtualization and optimization. He is a member of ACM and has guest edited several special issues in IEEE and Springer Journals. Currently, he is holding a visiting professor position at the University of Essex.

Haiyang Wang is an assistant professor in the Department of Computer Science at the University of Minnesota Duluth, MN, US. His research interests include cloud computing, peer-to-peer networking, social networking, big data and multimedia communications.

Jiangchuan Liu is currently an associate professor at the School of Computing Science, Simon Fraser University, British Columbia, Canada, and was an assistant professor in the Department of Computer Science and Engineering at The Chinese University of Hong Kong from 2003 to 2004. His research interests include multimedia systems and networks, wireless ad hoc and sensor networks, and peer-to-peer and overlay networks. He is a senior member of IEEE, member of Sigma Xi, an associate editor of IEEE Transactions on Multimedia, editor of IEEE Communications Surveys and Tutorials, and area editor of Computer Communications. He is also the TPC Vice Chair for Information Systems of IEEE INFOCOM'2011.

Song Lin received his B.S degree and Ph.D. degree from Tsinghua University, Beijing, China, in 2007 and 2012, respectively. His main interests include next generation Internet and P2P network.

Lei Xu received his B.S degree in computer science from Beijing Institute of Technology, China in 2006. He is working toward his master's degree in the Department of Computer Science & Technology at Tsinghua University, supervised by Prof. Yong Jiang. His research interests include datacenter networking and software-defined networking.