iPASS: Incentivized Peer-assisted System for Asynchronous Streaming

Chao Liang[†], Zhenghua Fu[‡], Yong Liu[†], and Chai Wah Wu[‡]
[†]Polytechnic Institute of NYU, Brooklyn, NY, USA 11201
[‡]IBM T.J.Watson Research Center, Hawthorne, NY, USA 10532
Email: cliang@photon.poly.edu, zfu@us.ibm.com, yongliu@poly.edu, cwwu@us.ibm.com

***Abstract*— As an efficient distribution mechanism, peer-to-peer technology has become a tremendously attractive solution to offload servers in large scale video streaming applications. However, in providing on-demand asynchronous streaming services, P2P streaming design faces two major challenges: how to schedule efficient video sharing between peers with asynchronous playback progresses? how to provide incentives for peers to contribute their resources to achieve a high level of system-wide Quality-of-Experience (QoE)? In this paper, we present iPASS, a novel mesh-based P2P VoD system, to address these challenges. Specifically, iPASS adopts a dynamic buffering-progress-based peering strategy to achieve high peer bandwidth utilization with low system maintenance cost. To provide incentives for peer uploading, iPASS employs a differentiated pre-fetching design that enables peers with higher contribution pre-fetch content at higher speed. A distributed adaptive taxation algorithm is developed to balance the system-wide QoE and service differentiations among heterogeneous peers. To assess the performance of iPASS, we built a detailed packet-level P2P VoD simulator and conducted extensive simulations. It was demonstrated that iPASS can completely offload server when the average peer upload bandwidth is 1.2 more times the streaming rate. Furthermore, we showed that the distributed incentive algorithm motivates peers to contribute and collectively achieve a high level of QoE.**

I. Introduction

Video-on-Demand (VoD) services enable users to watch their favorite videos at their convenient time. YouTube, an extremely popular VoD application on the Internet, serves 100 million distinct videos daily [1]. Traditional VoD solutions employ video servers and content distribution network (CDN) to stream video to viewers. The infrastructure cost grows linearly with user population and video quality. It will become very expensive for YouTube to stream higher resolution videos with TV or even high-definition quality. On the other hand, P2P technology utilizes resources available on peers and effectively offloads servers in large scale content distribution, such as file sharing [2] and live video streaming [3]. Recently, P2P technology has also been adopted to provide VoD services. In providing VoD services, P2P streaming design faces two major challenges: how to schedule efficient video sharing between peers with asynchronous playback progresses? how to provide incentives for peers to contribute their resources to achieve a high level of system-wide Quality-of-Experience (QoE)?

To address the asynchronous user playback issue, the *Cache-and-Relay* approach has been proposed. Peers store downloaded video in memory or hard disk, and relay the cached video to other peers in future, leading to asynchronous P2P video sharing. Early Cache-and-Relay based systems assume a small amount of video cache on peers and exploit asynchronous sharing between peers with close playback progresses. Through batching, peers are organized into groups according to their playback time and a tree-like topology is formed for peers in the same group to exchange video [4], [5]. Unfortunately, small video caching results in low P2P sharing efficiency. The structured P2P topology incurs high management overhead and is vulnerable to dynamic peer arrivals and departures. The recent advances in computer hardware technology make low-priced computers increasingly equipped with abundant memory and storage. New P2P VoD systems fully exploit the largely improved peer video caching capability for higher P2P sharing efficiency. In [6], [7], peers are effectively turned into distributed “video seeds” by caching a large volume of video clips on their hard disks. Longer video caching also makes it possible for P2P VoD systems to adopt mesh-based topology. Mesh-topology is robust to peer churn and easy to manage. It has demonstrated its success in many large scale file sharing [2] and live streaming systems [3]. Inspired by the successes of mesh, several mesh-based P2P VoD systems have been proposed [8], [9], [10]. In those systems, peers form one or multiple meshes randomly and exchange data with neighbors. Unlike in file sharing, in VoD systems, data sharing between peers are commonly uni-directional. Data flows from a peer to its neighbors with smaller playback progresses. We will show that random peering leads to poor peer resource utilization under this data flow directionality. How to design P2P VoD systems with high peer bandwidth utilization and low maintenance cost remains to be a challenging research problem.

Providing incentives for peers to contribute their resources is an essential design component for P2P systems in general. In file sharing systems, peers are motivated to upload to other peers in order to achieve a higher download rate from the system. By employing the *tit-for-tat* policy, BitTorrent punishes free-riders who do not contribute bandwidth to the system. In live streaming systems, peers are motivated to contribute more in order to get better playback quality. It was proposed in [11], [12] that, with scalable video coding, peers uploading more will be rewarded with higher video quality. Due to the asynchronous peer playback progress and the data flow directionality, tit-for-tat type of direct reciprocity incentive mechanism is not applicable in P2P VoD systems. In addition, to maintain the playback continuity, each peer needs to download video data before their playback deadlines. It is critical to design incentive mechanism for P2P VoD systems to balance the system-wide QoE and service differentiations among heterogeneous peers.

In this paper, we present iPASS, a novel mesh-based P2P VoD system, to address the previously described efficiency and incentive issues. iPASS adopts a dynamic Buffering-Progress-

Based (BPB) peering strategy to achieve high peer bandwidth utilization with low system maintenance cost. To provide incentives for peer uploading, iPASS employs a differentiated pre-fetching design that enables peers with higher contribution pre-fetch content at higher speed. A distributed adaptive taxation algorithm is developed to balance the system-wide QoE and service differentiations among heterogeneous peers. The contribution of this paper is three-fold:

1) We analytically study the impact of asynchronous peer playback progresses on the efficiency of mesh-based P2P sharing. We propose a distributed BPB peering strategy. Through analysis and simulation, we show that, with BPB peering, it is possible to achieve high peer bandwidth utilization, low maintenance cost and high peer churn robustness in mesh-based P2P VoD systems.
2) To the best of our knowledge, we are the first to to use differentiated pre-fetching as an incentive mechanism to motivate capable peers to contribute in P2P VoD system. We demonstrate that pre-fetchings on peers can be coordinated by an adaptive taxation algorithm to simultaneously maintain system-wide QoE and provide service differentiations among peers with different contributions.
3) To assess the performance of iPASS, we built a detailed packet-level P2P VoD simulator and conducted extensive simulations. Compared with previous P2P streaming simulators, our simulator simulates packet-level details. In addition, it can prolong the simulation duration to hours in order to study long-term system behaviors under a rich set of simulated scenarios.

The remaining of this paper is organized as follows. We briefly discuss the related work in Section II. The main design components are outlined in Section III. The detailed system implementation is presented in Section IV. The simulation setting and numerical results are presented in Section V. The paper is concluded in Section VI.

II. Related work

P2P sharing can greatly reduce server bandwidth cost to provide on-demand streaming service [13]. Early P2P VoD systems adopt structured streaming topologies and require delicate system management. P2Cast [5] groups peers according to their arrival time. Peers in the same group are organized into a multicast tree. Peers retrieve video content through a combination of streaming along the tree and patching from peers who arrived earlier. dPAM [4] employs distributed pre-fetching to improve system performance. oStream [14] constructs media distribution trees at the application layer to realize asynchronous media delivery. Recent advances in computer hardware technology largely improve peers' video caching capability and broaden the design spaces of VoD systems. Influenced by P2P file sharing systems, mesh-based data swarming has been adopted by new P2P VoD systems. BiTos [10] customized the Bittorrent protocol for on-demand video streaming. PONDER [9] divides video into multiple sub-clips and forms multiple meshes, one for each sub-clip. Peer selection and measurement based admission control was proposed to manage swarms. BASS [15] combines streaming from the server with Bittorrent-assisted downloading. The authors of [8] discussed the impact of segment scheduling, overlay management and network coding on the performance of swarming-based VoD systems. To combat free-riders, more and more attentions have been given to the design of incentive mechanisms in P2P streaming systems. A *tit-for-tat* type of substream trading algorithm was developed in [12] to provide incentive in live streaming systems with layered video coding. Authors of [16] proposed a taxation scheme to improve the overall social welfare through subsidizing resource-poor peers by exploiting resource-rich peers. The work in [11] utilizes a similar taxation scheme. Peers with more contribution join more substream trees to get better quality. However, providing incentive in on-demand systems remains a challenging problem.

III. iPASS: Design Overview

In this section, we present the two major design components of iPASS: *Buffering-Progress-Based (BPB) peering* and *Adaptive-Taxation-Based (ATB) pre-fetching*.

A. Modeling of Swarming-based Peer-assisted VoD System

In *peer-assisted* VoD systems, servers host publishers' videos and stream them to peers upon requests. To save bandwidth consumption on servers, peers viewing the same video form a P2P overlay network and redistribute videos among themselves. Severs are responsible for maintaining peers' playback continuity. If a peer cannot download video data from other peers before the playback deadline, it will download the missing data from the server directly, consequently, increase server bandwidth cost. A key design issue of P2P VoD systems is to minimize the server bandwidth cost by efficiently utilizing peers' upload bandwidth. P2P VoD systems have two unique features: the playback progresses on peers are *asynchronous*; peers can download content beyond its current playback range. In addition, to cope with bandwidth variations and peer churn, a peer normally buffers a certain amount of video beyond its playback progress.

1) Notations: To model a typical P2P VoD system, we introduce the following notations for peer i in the system:

- *Playback progress* p_i: the current playback position of peer i, indexed by the sequence number of the video chunk being played.
- *Buffering progress* b_i: the sequence number of the first missing chunk beyond current playback position p_i.
- *Buffering level* τ_i: the number of continuous buffered chunks beyond the current playback progress point. By definition, $\tau_i = b_i - p_i$.
- *Playback buffering threshold* w_{rd}: the number of buffered chunks necessary for smoothing playback. We call the sliding window $[p_i, p_i + w_{rd}]$ peer i's continuous playback range.
- *Contribution level* c_i: the number of chunks that peer i has uploaded to other peers since it joins the system.

Fig. 1 illustrates two different peer buffer statuses. On Peer 1, the buffer level τ_1 is lower than the playback buffering threshold w_{rd}. It is downloading the missing chunks in the continuous playback range. We call peer 1 is in the *normal playback mode*. On peer 2, the buffer level τ_2 is higher than the playback buffering threshold w_{rd}. Peer 2 is downloading chunks outside of the playback range. We call peer 2 is in the *pre-fetch mode*.

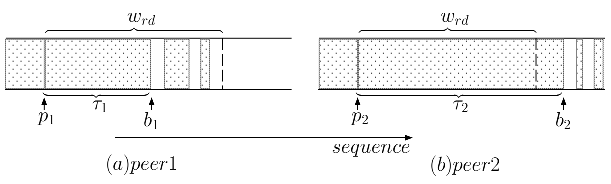


Fig. 1. Peer buffer status

Peers are assumed to have enough storage to cache what they ever playbacked. In terms of copyright issue, the content in the cache will disappear once the belonging peer quits the application (Similar as YouTube), and cannot serve as seed when it logins in the system next time. We also assume peers are not *strategic* but obedient to unveil their truthful information to each other.

2) Impact of Asynchronous Playback on P2P Sharing: First we investigate the impact of asynchronous peer playback on the efficiency of content sharing among peers. Let's start with a P2P VoD system with homogenous N peers, each of them with upload bandwidth u. Suppose each peer randomly selects k peers as its neighbors. The video length is L. At a given time instant, we assume peers' playback progresses are uniformly distributed among $[0, L]$. Peers store the content they have already played. Obviously, a peer can only serve peers with playback progress behind him. In addition, a peer divides its upload bandwidth equally to all its receivers.

Proposition 1: The expected download rate of a peer with playback progress x from other peers can be approximated by $u(lnL - lnx) + O(u)$.

Proof: Suppose the random variable X denotes selected neighbor playback progress and is uniformly distributed among $[0, L]$. Since peer selects k neighbors randomly and independently, X_i are independent for $i = 1 \dots k$. Then u_{Xx} = $\{$the download rate from user with playback progress X of its receivers$|X > x\}$, given only the peer with larger playback progress can be the supplier. Assume peer divides its upload bandwidth equally on all its receive neighbors, with $P(X < x) = x/L$, the download rate from user with playback progress x for its receiver can be approximated by $E(u'_x) = u/(k * P(X < x)) = uL/kx$. For peer x, then $E(u_{Xx}|X > x) = \frac{E(u'_{x+1})+E(u'_{x+2})+\ldots+E(u'_L)}{L-x} = \frac{\frac{uL}{k(x+1)}+\frac{uL}{k(x+2)}+\ldots+\frac{uL}{kL}}{L-x} = \frac{uL/k(lnL-ln(x)+O(1))}{L-x}$. Therefore we have $E(d_x^{max}) = \sum_k u_{X_k x} = k * P(y > x)E(u_{yx}|y > x) = u(lnL - lnx) + O(u)$ ■

The above proposition shows that the expected possible download rate drops logarithmically as the playback progress increases. For peers with larger playback progress, due to the random neighbor selection, they will find in their random neighbor set fewer suppliers from which they can download video from. In addition, a supplier with larger progress will be able to serve more download requests. Due to the equal bandwidth sharing, it will upload to each of its receivers at lower rate. These two factors conspire and lead to low download rates for peers with large playback progress. This shows that random peering and equal bandwidth sharing lead to low P2P bandwidth sharing efficiency.

B. Buffering Progress Based Peering

The bandwidth sharing efficiency in P2P systems is mainly determined by two factors: how peers are connected and how a peer allocates its upload bandwidth to all its neighbors. The former one decides how best the latter one can make with limited peering degree. In the previous section, we have demonstrated that random peering and equal bandwidth sharing is not efficient for asynchronous P2P VoD systems. Peers with larger playback progress have less opportunity to download from the P2P network. Intuitively, to increase the download rate of peers with large progress, the upload from peers close to the end of the streaming session should not be invested to peers just joined the session. More generally, we propose the Buffering Progress Based(**BPB**) peering to let peers connect to peers with close buffering progress. Peers form one structured mesh overlay with BPB peering strategy, instead of forming multiple sessions in patching[5] by grouping peers according to arrival time within certain threshold. In the mesh topology constructed under BPB, peers with similar playback progresses are strongly connected. Parts of the peers are suppliers with larger buffering progress. Parts of them are receivers with buffering progress lagging behind. And parts of them have very close progress and overlapping download interests, they may act as either supplier or receiver. On top of the BPB mesh, peers adaptively allocate their upload bandwidth to their neighbors to maximally reduce the complementary streaming requests to servers.

We formulate the following Linear Programming model to study the impact of peering and bandwidth allocation on server bandwidth. For peer i, let $nb(i)$ be its neighbor set, and u_{ji} be the download rate from peer j. The aggregate download rate from all its neighbors is $\sum_{j\in nb(i)} u_{ji}$, and then the complementary streaming rate needed from the server is $\max(0, r - \sum_{j\in nb(i)} u_{ji})$. The goal is to find the optimal peer bandwidth allocation to minimize the aggregate server cost.

$$\min_{\{u_{ji}\}} \sum_{i\in V}(r - \sum_{j\in nb(i)} u_{ji}) \tag{1}$$

$$\sum_{j\in nb(i)} u_{ij} \le u_i, \qquad i \in V \tag{2}$$

$$u_{ij} \le I_{ij} u_i, \qquad i, j \in V \tag{3}$$

$$\sum_{j\in nb(i)} u_{ji} \le r, \qquad i \in V \tag{4}$$

In the above formulation, I_{ij} denotes the buffering progress relationship between peer i and j, $I_{ij} = 1$ when $p_i > p_j$, otherwise equal to 0. Eq. (2) states the bandwidth constraint for each peer respectively. And Eq. (3) shows the content constraint among peers. Eq. (4) states the download speed constraint without pre-fetching.

The above optimal bandwidth allocation formulation is for general peering topology. We now use it to compare the server bandwidth saving of random peering and BPB peering. Towards this goal, we generate an instance of a peer-assisted video-on-demand system using a discrete simulation. During the simulated session with duration $T = 100$, peers arrive at the system according to a Poisson process with rate $\lambda = 2$. Peers stay in the system in a linear viewing manner till they finish the entire video viewing. The video rate is r and we assume all peers' download bandwidth is greater than r. There are two types of peers with upload bandwidth $1.5r$ and $0.5r$ respectively. The normalized average peer upload bandwidth is $\rho = \bar{u}/r = 1.2$.

With random peering, upon arrival, a peer randomly picks k peers already in the system as its neighbors. With BPB peering, peers are firstly ordered in the increasing order of their arrival times. A peer who arrived at the system with rank i will randomly pick k neighbors from peers with arrival ranks in the range of $[i - \delta * N, i]$ given total N online peers. By changing δ, we manipulate the playback progress closeness of neighbors in the constructed BPB graph. We then compare the server cost under BPR and random peering strategies under five snapshots of the system. For each snapshot, we solve the optimal bandwidth allocation problem defined in (1). Figure 2 shows the minimum server cost can be achieved

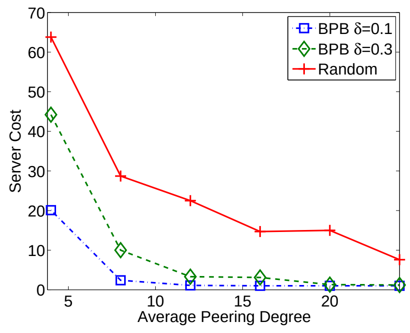


Fig. 2. Server Cost with $\rho = 1.2$

with different peering strategies. The results indicate that with limited peering degree, BPR-peering can significantly reduce the server cost compared with random peering. The results obtained here is only the lower bounds on the server bandwidth cost. In Section V, we will compare the server bandwidth saving of random peering and BPB peering through detailed packet-level simulations.

C. Adaptive Taxation Based Pre-fetching

To maintain their playback continuity in face of peer churn and network dynamics in P2P video systems, peers normally buffer certain amount of data ahead of the playback progress. Furthermore, in P2P VoD systems, peers with high download rate can *pre-fetch* content beyond their playback points and potentially become *seeds*, namely, nodes with the whole content, long before their playback ends. From the system point of view, more seeds in the system, more efficient the content sharing among peers. As will be shown through simulations in Section V : *with large enough peer bandwidth resource and high scheduling efficiency, the seeds that evolved from regular peers with pre-fetching may completely take the place of the servers and results in zero server cost*. From individual peers point of view, with pre-fetched content in the buffer, they can enjoy smooth non-linear viewing operations, such as fast-forwarding and jumping. Moreover, peers can finish the download process of the whole content before they finish the playback, and they have options to leave the system to proceed other Internet applications without interference.

Providing incentive in asynchronous VoD system is challenging. The asymmetric data flows between peers with different playback progresses make direct reciprocity incentive mechanisms, such as tit-for-tat of BitTorrent, infeasible. In our design, we use pre-fetching as an incentive to motivate peers to contribute more to obtain higher download rate from the system. To coordinate the asynchronous demands of peers and maintain system-wide Quality of Experience (QoE), we propose a *Adaptive Taxation* scheme to regulate the pre-fetching on heterogeneous peers. Original taxation scheme[16] is applied to provide incentive in live streaming system. The bandwidth can be regarded as peer's *wealth*. Resource-rich peers contribute more bandwidth to the system, and subsidize for the resource-poor peers. The tax regulated redistribution of peer wealth helps improve the social welfare and then reduce server cost. The tax ratio is fixed in the original scheme. To achieve budget balanced, the *demogrant* (i.e., one peer who does not contribute anything, still receives the demogrant rate) rate is adaptive. However, the situations differ in adaptive taxation scheme. Instead of differentiated playback quality, peers in peer-assisted system differ from download rate and have base playback rate guaranteed. Therefore in our adaptive taxation method, the demogrant rate is fixed to be equal to the playback rate and tax ratio should be adaptive. Suppose we pose a taxation ratio t on peers. Then one peer with contribution level c_i and lifetime T_i, could get the average download rate r_i to accumulate expected buffer level

$$\bar{\tau}_i = (r_i - r)T_i = \frac{c_i}{t}. \tag{5}$$

To make the aggregate tax revenue $\sum r_i$ and budget expenditure $\sum c_i/T_i$ balanced, the taxation ratio t needs to be adaptive to the system wide resource availability. To decide the ratio, we have

$$t = \sum c_i / \sum \tau_i \tag{6}$$

. In a resource rich system, peers accumulate different amount of buffering levels proportional to their contributions and the system tax rate t. In a resource deficit system with small peer average bandwidth $\bar{u} < r$, peers bandwidth are not enough to sustain their normal playback demands and needs help from the server. In this case, it could be difficult for any peer to accumulate large buffering level and $t \to \infty$, then peers try to fetch chunks in the continuous playback range. How to adapt t with the system resource is crucial in the adaptive taxation scheme. Due to peer dynamics and resource imbalance, it could be infeasible to tackle the issue in a centralized manner. A distribution protocol with the adaptive taxation is introduced in the following sections.

IV. iPASS: System Design

In this section we present the detailed design of iPASS system. We focus on the implementation of BPB peering and ATB pre-fetching.

A. Architecture

Similar to most deployed large scale P2P streaming systems, iPASS employs a *tracker* to keep track of peer arrivals and departures. The tracker maintains a list of active peers in the system. When a new peer joins in, it first contacts the tracker for an initial peer list. Then new peer makes connections to peers on the returned list and starts to exchange signaling information and video data with them. Through signaling, peers exchange with their neighbors information about their buffering progresses, contribution levels and neighbor lists. iPASS adopts the pull based data exchange mechanism. A peer pulls video chunks from its neighbors by sending download requests. To avoid contention due to uncoordinated requests to the same peer, we introduce *pull tokens* for peers. Each peer periodically sends out pull tokens to its neighbors to give them permissions to pull chunks from him. The total number of tokens that one peer sends out is determined by the number of chunks that it can serve in each round. The number of tokens that a peer sends to a neighbor is determined by the contribution level of the neighbor, and is calculated by a distributed implementation of the ATB pre-fetching algorithm described in the previous section. Due to asynchronous pre-fetching, a peer may become out-of-sync with its neighbors. If so, to maintain the BPB peering, it needs to change its neighbors. A peer will find new neighbors by querying the tracker or searching through its neighbors' neighbor lists. For example, idle seeds and peers lacking enough number of suppliers may turn to find complementary neighbors.

B. BPB Peering Implementation

The key to BPB peering is to find peers with close buffering progresses. To facilitate BPB peering, the tracker sorts the list of active peers according to their arrival times. When a new peer joins in, the tracker records its arrival time and append it to the end of peer list. Then the tracker will return the new peer with an initial peer list consisting of a number of random peers at the end of the list. Those peers will be the suppliers for the new peer.

When there is no pre-fetching, buffering on peers advances roughly at the same pace, namely the playback rate. Peers who arrive close in time will remain close in buffering progress. During the session, when a peer needs to connect to new neighbors, either due to neighbor departures or unsatisfactory peering connections, it can contact the tracker for additional peers. The tracker can quickly search through the sorted list to find peers with close buffering progress for the requesting peer. In addition, due to BPB peering, a peer's neighbors' neighbors should also have close buffering progresses with the peer. Without going to the tracker, a peer can find new "close" neighbors in the neighbor lists returned by its neighbors.

With pre-fetching, buffering on peers advance at different rates. A peer joins the system later can possibly download video faster than his neighbors who arrived earlier and gain larger buffering progress. Once this happens, the download rate of the peer will be slowed down due to the lack of enough suppliers. The peer should then trigger *dynamic BPB peering* to find more suppliers satisfying the BPB peering criterion. Fig. 3 shows a simplified example of dynamic BPB peering. Towards the goal of downloading the whole video, node n_a runs on the "express track" with larger download speed, while its neighbors runs on the "local track" with smaller download speed. As time evolves, it catches up with the buffering progress of its neighbors. To maintain its download rate, it connects with n_1 with larger buffering progress and disconnects from peer n_5 with the smallest buffering progress.

To facilitate this dynamic BPB peering, a centralized solution is to have the tracker keep track of peers' buffering progresses and help peers to find new neighbors with close buffering progresses. Peers need to periodically report their current buffering progresses to the tracker. And the tracker also needs to constantly resort the peer list. This will incur large signaling and processing overhead on the tracker and peers. On the other hand, peers constantly exchange their buffering progresses with their neighbors. Due to dynamic BPB buffering, there is a good chance that a peer, even doing fast pre-fetching, can find peers ahead of him by searching through the neighbor lists returned by its neighbors. Then instead of requesting from the tracker, peers can request complementary peer lists from neighbors and pick appropriate peers with close buffering progress to connect.

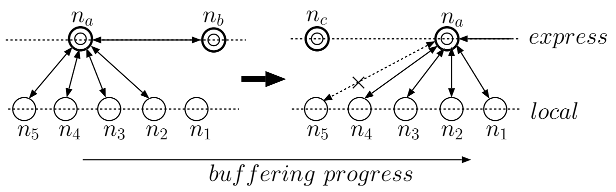


Fig. 3. Dynamic BPB peering

C. Signaling between Neighbors

In iPASS, peers need to frequently collect information from their neighbors and exchange data availability using *buffer-map*. A buffer-map of a peer consists of a sequence of binary bits, each of which indicates the availability of one specific chunk on that peer. In live P2P streaming systems, due to the synchronous peer playback, at any time instant, the chunks that peers need to download fall into a small moving window covering several minutes worth of video. The buffer-map length can be kept short even though the chunk size is chosen to be small (tens of KiloBytes).[1] In P2P file sharing, peers randomly download different portions of files. Buffer-maps have to indicate the data availability for the whole file. To limit the signaling overhead, large chunk sizes (hundreds of KBs) are chosen to reduce the size of bitmap. Similar to file sharing, peers in VoD systems are asynchronous. Similar to live streaming, VoD systems need to maintain the continuous playback on peers. It is challenging to design VoD buffer-map to simultaneously achieve high utilization and low system overhead.

To address this issue, we define the *interested area* of a peer as the range of chunks that the peer is currently downloading. In the normal playback mode, peer i needs to retrieve chunks in their current playback range $[p_i, p_i + w_{rd}]$. Once all chunks in the playback range have been retrieved, it enters into the pre-fetching mode and starts to download chunks falling into its pre-fetching window. Therefore, the interested area of a peer is either its current playback range, if it is in the normal playback mode, or the pre-fetching window, if it is in the pre-fetching mode. Peers generate buffer-maps only for chunks in the interested area. Furthermore, peers could only send the buffer map to neighbors who have overlapping interested area in order to reduce overhead. In addition, peer i send to its

[1]Small chunks reduce content bottleneck and improve peer bandwidth utilization.

neighbors information on its buffering point b_i, buffering level τ_i, contribution level c_i.

D. Chunk Scheduling between Neighbors

Chunk scheduling determines the data flows among neighbors. iPASS employs pull-based approach. Peer needs to handle the token distribution and pull requests among receivers, denoted by the set ψ, consisting of peers with interested areas either overlapped or totally covered by chunks already buffered by this peer. After obtaining buffer-maps from its neighbors, a peer sends pull requests to download missing video chunks from its neighbors who have them. Due to distributed scheduling, peer i may receive multiple pull requests from peers in its receiver set $\psi(i)$. Some of the requests will be delayed or even disposed if peer i cannot fulfill all of them in time. To avoid contention, we introduce tokens to regulate pull requests send to a peer. Specifically, peer i periodically sends tokens to peers in set $\psi(i)$ to give them permission to pull data from him. The number of tokens that peer i sends is determined by how many chunks it can serve within each round. In the strategy without pre-fetching, the tokens of peer i is randomly distributed to peers in $\psi(i)$. In the pre-fetching mode, the token distribution should be conducted to maintain normal playback on all peers and enable differentiated pre-fetching based on peers' contribution. The Adaptive Taxation Based pre-fetching algorithm described in Section III-C is an ideal centralized solution, cannot be implemented in a large system. We developed a distributed token distribution algorithm to realize ATB pre-fetching.

Algorithm 1: ATB Token Distribution on Peer i

```
1 input: ${\tau_k, c_k, \forall k \in \psi(i)}$
2 output: $P(k)$: fraction of tokens to peer $k$
3 $t \leftarrow \sum_{k\in\psi(i)} \tau_k / \sum_{k\in\psi(i)} c_k$
4 for $k \in \psi(i)$ do
5     if $\tau_k \leq w_{rd}$ then $e_k \leftarrow \max(w_{rd}, c_k/t) - \tau_k + 1$
6     else $e_k \leftarrow max(c_k/t - \tau_k, 1)$
7     $sum \leftarrow sum + e_k$
8 end
9 for $k \in \psi(i)$ do $P(k) \leftarrow e_k/sum$
```

The ATB token distribution algorithm is presented in Algorithm 1. Instead of assessing a universal tax ratio based on global information, peers deduce it locally based on information from their neighbors. The tax ratio t calculated by peer i is the ratio between the aggregate buffering levels and the aggregate contribution levels of peer i's neighbors. The target buffering level $\bar{\tau}_k$ of a neighbor k is its contribution level c_k divided by t. Then peer i determines the *expected tokens* e_k to peer k as $\bar{\tau}_k - \tau_k$. ATB scheduling gives neighbors in the normal playback mode priority in access tokens. If a neighbor k's buffer progress τ_k falls behind the playback buffering threshold w_{rd}, peer i will give at least $w_{rd} - \tau_k$ tokens to peer k so that it can download chunks in the playback range. After calculating e_k for all its neighbors, the peer can determine the fraction of tokens for each neighbor in this round and then assign tokens according to the distribution. Fig. 4 illustrates an example of ATB token distribution. w_{rd} is set to 1. Peer n has five neighbors and only n_1, n_2 and n_3 are its receivers.

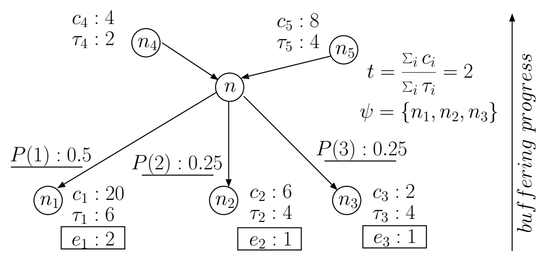


Fig. 4. Illustration of token distribution

First the tax ratio are calculated to be $t = 2$. Then based on the buffering level and contribution level, the expected number of tokens e_k are calculated for each peer. Finally the fractions of tokens sent to n_1, n_2 and n_3 are decided as 0.5, 0.25 and 0.25 respectively.

After a peer receives pull tokens from all its neighbors, it will decide from which neighbor to pull which chunk. Various chunk requesting algorithms in live streaming, such as rarest-first or oldest-first, can be applied. However, their effects could be skewed with asymmetric data flow direction due to asynchronous buffering progress of peers. In a simplified manner, one can request missing chunks randomly from the neighbors which hold the chunk and also send the token. Tokens from a neighbor will be disposed if the peer does not send pull request to that neighbor in this round. This is to avoid disturbances to the efficiency of scheduling in future rounds.

V. Performance Evaluation

We use simulations to evaluate the performance of the proposed peering and pre-fetching strategies. *bpbp_np* and *ranp_np* refer to the BPB-peering and random peering strategies without pre-fetching respectively. *bpbp_inc* refers to our iPASS strategy, the combination of the BPB-peering with ATB pre-fetching. A random peering strategy with pre-fetching, denoted by *ranp_wp*, is also developed to make the comparison comprehensive.

A. Simulation Setup

We developed a packet-level event-driven simulator in C++ to study the performance. Our simulator adopts the infrastructure of the simulator engine of [17] simulating the end-to-end latency in terms of real-world latency measurement results. Two 4-CPU servers are applied to accelerate the simulations.

We follow the common consumption that peer download bandwidth is large enough and bottlenecks happen only at the edges of the network. There are three DSL types of nodes with bandwidth 1Mbps, 384kbps and 128kbps. The video streaming rate is 400kbps and each chunk has 5 KB size. We vary the distributions of these nodes to adjust the normalized peer average bandwidth, as shown in Table I. In the simulation, we use a single video with 30mins length. One single simulation round lasts for 90mins to get a better view of the system behavior. We believe that the video length and the simulation duration are already long enough to demonstrate the features of different strategies. The peer arrivals follow the Poisson process with arriving rate $\lambda = 1/4$ per second. The number of online peers maintains constantly around 500 after the startup phase and there are around $1,500$ peers joining the system during the whole session. The default

TABLE I

NORMALIZED PEER AVERAGE BW AND THE CORRESPONDING FRACTION OF PEER TYPES

ρ	Fraction of Peers (1M,384k,128k)	ρ	Fraction of Peers (1M,384k,128k)
0.90	$0.15, 0.39, 0.46$	1.40	$0.34, 0.52, 0.14$
1.00	$0.20, 0.40, 0.40$	1.50	$0.43, 0.38, 0.19$
1.12	$0.23, 0.46, 0.31$	1.60	$0.49, 0.36, 0.15$
1.20	$0.25, 0.53, 0.22$	1.70	$0.54, 0.32, 0.14$
1.30	$0.30, 0.50, 0.20$	1.80	$0.60, 0.30, 0.10$

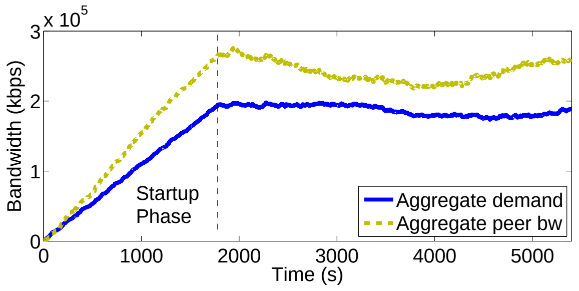
number of neighbors of each peer is 15. The size of the playback buffering threshold and pre-fetching window are both 4 seconds. Peers broadcast buffer-map messages every 0.5 second and the token number information is piggybacked within the message. The server bandwidth cost consists of two parts, due to the complementary pull from peer for missing chunks and request scheduled from peers who receive the tokens from server respectively. The number of tokens sent out periodically from server corresponds to 1Mbps. To make the comparison fair, we generate the peer arrivals and upload bandwidth configuration beforehand and use the same setting to compare different strategies.

B. Numerical Results

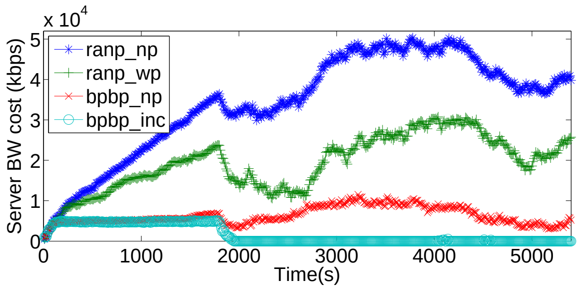
We first show the performance of various strategies on server bandwidth saving in the *linear viewing* mode. Peers will not leave the system before they finish the whole video playback. The results on differentiated pre-fetching are presented next. Then we study the performance with batch peer joins and early peer departures. At last, we compare the performance of iPASS with other P2P VoD systems.

1) Effectiveness on server cost saving: The server bandwidth saving is the most important performance metric to evaluate the different P2P strategies.

•**Server cost evolution illustration.** We begin by showing the



(a) Demand vs. resource



(b) Instant server cost

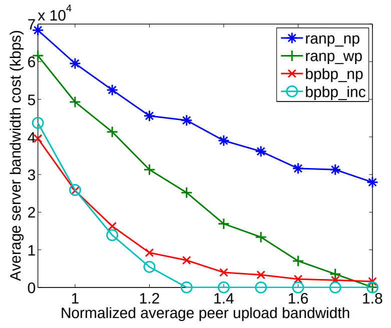
Fig. 5. Server cost under different peering strategy

evolution of server cost during one simulation session. Fig. 5(a) shows the instant aggregate user demand and the peer bandwidth when the normalized average peer bandwidth(ρ) equals to 1.3. There are no peers in the system at the beginning. The first peer finished playback and leave the system at $1,800$ second. The time period $[0, 1, 800]$ is the *system startup phase*. Fig. 5(b) presents the instant server cost under the different strategies. We can observe that the server cost of random peering strategies increase almost linearly at the startup phase as the number of peers increases, then the curves oscillate closely with the instant peer average bandwidth. However, for BPB-peering strategies, it is interesting to observe that the server cost increases in a short period and maintains almost constant at the startup phase. Peers join the system early have limited data to share with each other. The server has to stream data to them directly. When more peers get into the system, peers start to download data from each other. When the startup phase is over, the server cost drops nearly to zero in $bpbp\_inc$ strategy. Later simulation results show that a certain amount of peers evolve into seeds can take the place of the server. Without pre-fetching, $bpbp\_np$ is also sensitive to the average peer bandwidth resource. It successfully control the server cost at low level. We can find when $\rho = 1.3$, in the comparison of original streaming solution without P2P support, the random-peering without pre-fetching strategy ($ranp\_np$) can save at least around 75% server bandwidth. The saving can be improved to 85% with pre-fetching. With BPB-peering, the $bpbp\_np$ can enhance the saving further to around 95%. Moreover, $bpbp\_inc$ can sustain the system without server cost after the startup phase.

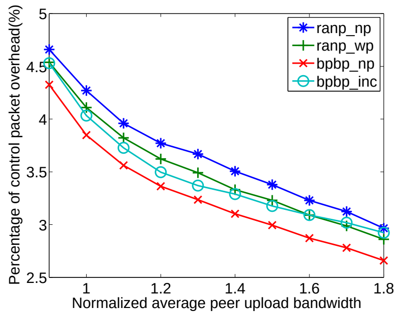
•**Performance with various peer distribution.** Next we examine the server cost savings with different normalized peer average bandwidth. Fig. 6(a) shows the average server cost after the first 50 mins. As the system resource increases, the cost of all strategies drops. $bpbp\_np$ and $bpbp\_inc$ both achieve most bandwidth saving. Especially $bpbp\_inc$ can sustain itself without server when $\rho > 1.2$. *The BPB-peering can effectively improve the scheduling efficiency, which results in more server bandwidth saving.* Pre-fetching enables peers to download future content with extra bandwidth, thus reduces the possibility of data pull from the server in the future. The $ranp\_wp$ strategy with pre-fetching can also work without server when $\rho = 1.8$. When the normalized average bandwidth is 0.9, $bpbp\_np$ slightly outperforms $bpbp\_inc$. We believe this is because that pre-fetching potentially impairs some peers' normal playback when the whole system is in a bandwidth resource deficit status. This disadvantage can be conquered in iPASS by giving more preference to neighbors who haven't fill up the playback window during the scheduling.

Although $bpbp\_np$ and $bpbp\_inc$ perform closely in terms of server bandwidth saving, pre-fetching of $bpbp\_inc$ produces seeds in the system. Fig. 6(c) illustrates the number of seeds during the simulation with normalized bandwidth equal to 1.5. It is very impressive that for $bpbp\_inc$ the seed number can even reach nearly 40% of all peers. On the other side, the ineffectiveness of random peering leads to fewer number of seeds in $ranp\_wp$. The existences of seeds make the system resource allocation more flexible and thus more robust to peer dynamics. Furthermore, seeds can completely take the place of the server.

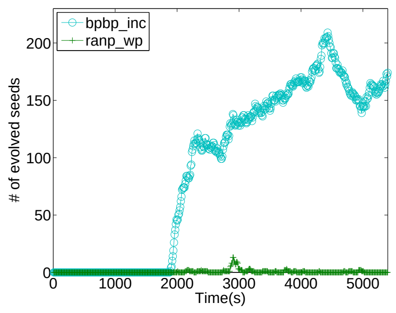
Peers only exchange the interested area information, which



(a) Average server cost in stable period

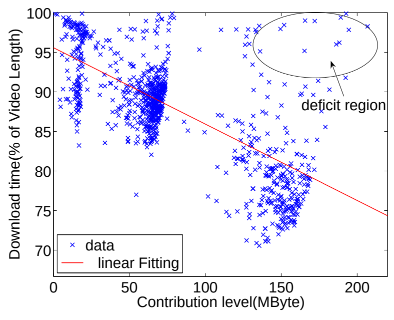


(b) Control traffic overhead

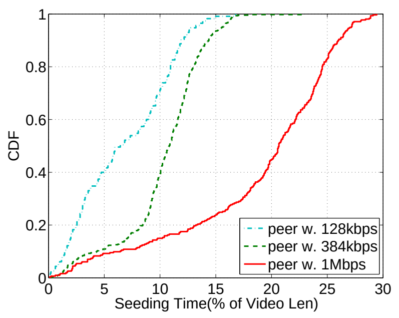


(c) Number of evolved seed

Fig. 6. Performance with various normalized peer bandwidth distribution



(a) Download time vs. contribution amount



(b) CDF of seed lifetime

Fig. 7. Performance with differentiated pre-fetching

is efficient to keep the overhead low. Fig.6(b) shows the control traffic throughput compared with data traffic. The overhead contributes less than 5% percentage for all cases. As the resource increases, the exchange between peers become more effective with large enough bandwidth, which leads to less control overhead in return. The same phenomena can be observed between random peeing and BPB-peering strategies, because the latter is more effective than the former.

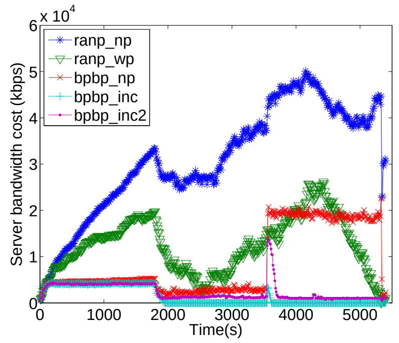
2) Impact of Differentiated Pre-fetching: Next we study the system performance with differentiated pre-fetching. Fig. 7(a) plots the correlation between peer's download rate and contribution level as $\rho = 1.4$. The crosses closely scatter along the linear fitting line, which indicates larger contribution peers can finish download faster. Peers in the deficit region are believed to be among the earliest batch of peers which can hardly find other suppliers to maintain the deserved download rate although they contribute a lot. As more and more peer become seeds, the download times of all peers decrease correspondingly. But the peers with larger contribution still finish sooner.

The contributions of peers are limited by their upload bandwidth. Fig. 7(b) plots the cumulative distribution of the *seeding time* of different types of peers, which is defined as the duration from the time a peer finishes video downloading till its departure. The peers with zero seeding time are not counted. We can observe that larger bandwidth peers get longer seeding time. Peers with 1Mbps bandwidth have average seeding time of 18.6% of the video length, while the average seeding time for peers with 384kbps and 128kbps are 9.9% and 6.3% respectively. Differentiated pre-fetching enlarges the seed capability further by encouraging peers with larger bandwidth to become seeds earlier.

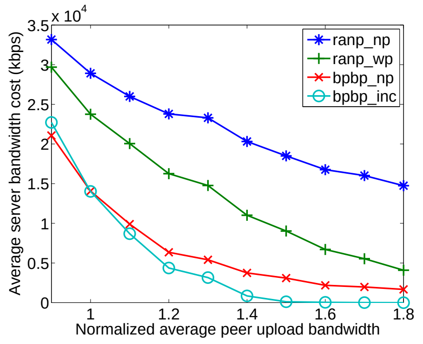
3) Batch Join & Early Departure Scenarios: Simulations in previous sections assume peers are in linear viewing mode and only leave the system after they finish their playback. We now study the system's performance under different peer churn models. We start with the flash-crowd scenario where a batch of peers joins the system at the same time. As $\rho = 1.5$, 100 peers, almost 20% of the maximum number of online peers, suddenly join the system simultaneously around $3,600$ second. Fig. 8(a) shows the server cost evolution. Pre-fetching prevents the bandwidth cost of $ranp\_wp$ from jumping up significantly. However, there are big jumps in both non-pre-fetching schemes $ranp\_np$ and $bpbp\_np$. At the same time, $bpbp\_inc$ is highly robust against batch peer arrivals. There is only a small pulse in the server cost after the batch arrival. The server cost quickly goes back to zero afterward.

Different from the linear viewing scenario, peers may also leave the system without finishing playback. We also study the impact on the system performance due to peer early departures. In this simulation, the peer lifetime follows a Weibull distribution. With Weibull distribution parameters $(1400, 4)$, peers leave the session gradually starting from the 271th second. And there are only 5.7% peers who will watch the whole video without early departure. Fig.8(b) shows the average server cost under various peer bandwidth distributions. The server cost of all strategies decreases almost half compared with Fig.6(a). This is because the number of simultaneously online peers decreases due to the early peer departures. But the relative performance order among different strategies remains similar. The $bpbp\_inc$ still achieves the best performance, and no server cost is needed when $\rho > 1.4$.

As an incentive in iPASS, peers are allowed to leave the system after they finish downloading the whole video. In that case, it cannot continue to stay as a seed to serve others. To



(a) Batch join scenario



(b) Early departure scenario

Fig. 8. Impact of batchjoin and early departure

verify the impact, we assume all peers are selfish and they will leave the system as soon as they finish the downloading. The system performance under this assumption is plotted in Fig. 8(a) using the curve denoted as $bpbp\_inc2$. The performance is close to $bpbp\_inc$. The peak value of the pulse due to batch join at around $3,600$ second is also less than $bpbp\_np$ and the pulse soon disappears as time evolves. The adaptive taxation scheme subsidizes large peers' bandwidth for small peers. Therefore the system still benefits a lot from the large peers' contribution before they finish the whole download.

4) Comparisons with other VoD Systems: Without detailed system implementation and parameter settings, we cannot conduct head-to-head comparisons between iPASS and other P2P VoD systems. We qualitatively compare iPASS with some known mesh-based P2P VoD systems using performance numbers reported by the authors. Simulations of BiTos[10] assume all users arrive at almost the same time. They are not comparable with common asynchronous peer simulation setting. In PONDER[9] system, the server cost saving can reach 95% when $\rho = 2.9$. In [8], even the best approach with network coding cannot let chunk delivery ratio exceed 70%. The saving in BASS[15] can only reach 34% with their own setting. The results are all simulated in linear viewing scenario. While in iPASS, the system can sustain itself without server when $\rho > 1.2$. Moreover, with early departure, the system can also sustain itself when $\rho > 1.4$ with only 5.7% linear viewing peers.

VI. Conclusion

In this paper, we present the design of iPASS, a novel mesh-based P2P VoD system. iPASS achieves high peer bandwidth utilization at low system maintenance cost by adopting a dynamic buffering-progress-based peering strategy. To provide incentives for peer uploading, iPASS employs a differentiated pre-fetching design that enables peers with higher contribution pre-fetch content at higher speed. We further demonstrated that pre-fetching on peers can be coordinated by an adaptive taxation algorithm to simultaneously maintain system-wide QoE and provide service differentiations among peers with different contributions. Through detailed packet-level simulations, we show that iPASS can efficiently offload server and achieve the desired balance between the system-wide QoE and service differentiations among heterogeneous peers.

As the next step, we will proceed to prototype iPASS and test its performance in real network environments. We are also interested in enhancing the BPB peering strategy to handle users' video seeking operations, such as fast-forwarding and skipping. The differentiated pre-fetching mechanism can be combined with scalable video coding to provide additional incentives for capable peers to contribute more. The adaptive taxation scheme will be redesigned to incorporate video quality differentiations.

References

[1] Youtube, "Youtube Homepage," http://www.youtube.com.
[2] "BitTorrent Homepage," http://www.bittorrent.com/.
[3] PPLive, "PPLive Homepage," http://www.pplive.com.
[4] A. SHARMA, A. BESTAVROS, and I. MATTA, "dPAM: A distributed prefetching protocol for scalable asynchronous multicast in p2p systems," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2005.
[5] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer Patching Scheme for VoD Service," in *World Wide Web Conference*, 2003.
[6] 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 *Proceedings of ACM SIGCOMM*, 2008.
[7] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "PushtoPeer VideoonDemand system: design and evaluation," in *IEEE Journal on Selected Areas in Communications*, 2008.
[8] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and D. Gunawardena, "Is High Quality VoD Feasible using P2P Swarming?" in *Proceedings of International World Wide Web Conference*, 2007.
[9] Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, "PONDER: Performance Aware P2P Video-on-Demand Service," in *Proceedings of GLOBECOM*, 2007.
[10] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications," in *IEEE Global Internet Symposium*, 2006.
[11] Y.-W. Sung, M. Bishop, and S. Rao, "Enabling Contribution Awareness in an Overlay Broadcasting System," in *Proceedings of ACM SIGCOMM*, 2006.
[12] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, and Y. Wang, "Substream Trading: Towards an Open P2P Live Streaming System," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2008.
[13] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-Demand be Profitable?" in *Proceedings of ACM SIGCOMM*, 2007.
[14] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," in *IEEE Journal on Selected Areas in Communications*, 2004.
[15] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "BASS: BitTorrent Assisted Streaming System for Video-on-Demand," in *International workshop on multimedia signal processing (MMSP)*, 2005.
[16] Y. hua Chu, J. Chuang, and H. Zhang, "A case for taxation in peer-to-peer streaming broadcast," in *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, 2004.
[17] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the power of pull-based streaming protocol: can we do better?" *IEEE Journal on Selected Areas in Communications*, 2007.