2011

Providing NPR-Style Time-Shifted Streaming in P2P Systems

Zhipeng Ouyang
University of Nebraska - Lincoln, zouyang@cse.unl.edu

Lisong Xu
University of Nebraska - Lincoln, xu@cse.unl.edu

Byrav Ramamurthy
University of Nebraska at Lincoln, bramamurthy2@unl.edu

2011 IFIP Wireless Days (WD), doi: 10.1109/WD.2011.6098196

Providing NPR-Style Time-Shifted Streaming in P2P Systems

Zhipeng Ouyang, Lisong Xu and Byrav Ramamurthy
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588-0115, U.S.A.
Email: {zouyang, xu, byrav}@cse.unl.edu

***Abstract*—Digital video recorder (DVR) style and non-prerecording (NPR) style are two possible implementations for P2P-based time-shifted streaming, but existing P2P streaming solutions are not suitable for implementing the NPR method. Since peers can view any arbitrary video segments which have been broadcasted, they might cause severe video quality problems and the server bandwidth consumption can become high. In this paper, we focus on minimizing the server bandwidth consumption to maintain smooth streaming service in NPR-style P2P-based time-shifted streaming. To reduce the server cost, peers prefetch segments which are not required for their current viewing. Hence even if they are viewing different parts of the video, they can exchange segments with one another. However, segment prefetching competes for bandwidth with ordinary segment fetching, and it might bring negative impact. A good prefetching solution should not affect a peer's viewing experience. We formulate the problem of finding a prefetching solution as an optimization problem, with the objective to minimize the server bandwidth consumption. Then we propose a heuristic algorithm by decomposing the global optimization problem into a set of smaller problems. Each peer runs this algorithm to determine which segments to prefetch and how to serve other peers. Simulation experiments demonstrate that our design provides P2P-based time-shifted streaming at low server bandwidth consumption.**

I. Introduction

Time-shifted streaming is an application which allows users to view live video content in a more convenient way. Users can view the video segments which have been broadcasted before they join the system. The service provider predetermines a time-shifted window to define how much past video a user can view. For example, when a peer joins the system, he/she can choose to view any video segments within the last 3 hours. Because of this flexibility, time-shifted streaming is very attractive. Although it has been implemented in typical television systems for a while [11], there is not much literature discussing P2P-based time-shifted streaming. In this paper, we address the design problem of P2P time-shifted streaming systems.

There are two options to implement P2P-based time-shifted streaming. One is digital video recorder (DVR) style, in which peers start the application at the beginning of the live broadcasting, and they use the recording function to save the video, so that they can view it later. In the other option, peers can start the application at any time during or even after the live broadcasting to view the video. It does not require recording before their viewing, and we called it non-prerecording (NPR) style. The DVR style can be implemented using existing live streaming solutions with minor changes. But the NPR style is challenging. As peers are viewing different video portions, they might not fetch the video segments in time and can happen glitches. Even if peers can request these segments from the streaming server, the server capacity is limited and the system scalability is affected.

A straightforward idea is prefetching, in which peers fetch the segments that they will view immediately as well as the segments that they will not view immediately. Any segments within the time-shifted window can be prefetched. Thus the segments are broadly distributed among all peers, and this increases the probability for peers to fetch segments from other peers.

On the other hand, prefetching implies more bandwidth consumption, and inappropriate prefetching would have negative impact on the system. For example, the segment prefetching requests compete for bandwidth with other regular segment fetching requests. Therefore, a good prefetching solution should efficiently use peers' spare bandwidth so that it does not impair the streaming quality and should not considerably increase the server load. We model the design of an efficient P2P-based time-shifted streaming into a prefetching optimization problem.

Our goal in this paper is to design a P2P-based time-shifted streaming system with low server bandwidth consumption. We first investigate possible time-shifted streaming implementations: DVR style and NPR style. Then we focus on the design of NPR-style time-shifted streaming. We formulate it as a problem of finding an optimal prefetching solution, so that peers are able to receive smooth video streams with low server bandwidth consumption. We set up a global optimization problem with constraints to ensure smooth video streams. Then we propose our distributed heuristic algorithm and compare its performance with the centralized optimal solution through simulations.

The remainder of the paper is structured as follows. We discuss the related work in Section II. Section III is the problem formulation and our design overview including the distributed algorithm. Simulation sample results are in Section IV. Finally, we conclude this paper in Section V.

978-1-4577-0638-7 /11/$26.00 ©2011 IEEE

	Time-Shifted Streaming	Style
PPStream	No	N/A
SopCast	No	N/A
PPTV	Yes	DVR
TVUPlayer	Yes	DVR
UUSee	Yes	NPR

TABLE I: Time-shifted deployment in P2P streaming

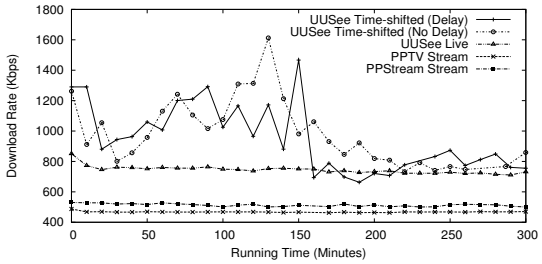

Fig. 1: Peers in time-shifted streaming of UUSee also download the segments not immediately viewed.

II. Related Work and Time-shifted Streaming in Commercial Systems

Generally, the P2P streaming systems have tree, multi-tree and mesh-based structures [5], in which the mesh-based structures are more reliable in case of peer churn. For example, [12] use mesh structures, and formulate P2P live streaming as a network flow problem to maximize the overall performance. Wang *et. al* [10][9] study cross-channel bandwidth allocation problems to improve streaming quality of all channels.

There are some studies in time-shifted streaming. In [3], time-shifted streaming is used to guarantee video continuity. Peers that suffer service disconnection can join a time-shifted stream. P2P-based time-shifted streaming also attracts some attention. [2] proposes that a peer uses one cache to store the content which they are watching, and another cache to store the video segments to implement the time-shifted feature. In [4][6], the time-shifted feature is implemented by caching a portion of the video stream at each peer and serving it asynchronously. In a TV system, a DVR records the video which is viewed at a time more convenient to users [7]. However, this option has a critical disadvantage: a peer is not able to view the past segments before it joins. UUSee [8] provides the NPR option so that peers can access the past parts before their joining times.

We investigate time-shifted streaming in popular commercial P2P streaming systems, and the results are summarized in Table I. PPTV and TVUPlayer already provide DVR-style time-shifted streaming service with which a peer can pause a video afer it joins the system. The lively broadcasted segments are downloaded to peers' local memory, and therefore peers are able to view any of them at any time. UUSee has NPR-style time-shifted streaming. To the best of our knowledge, only UUSee allows peers to access the video segments which have been broadcasted about 40 minutes before their joining times.

We have conducted experiments to measure three P2P streaming systems: PPStream, PPTV and UUSee. These three systems represent no time-shifted, DVR-style and NPR-style systems, respectively. We run these applications and record their corresponding download rates. In UUSee, the time-shifted streaming and the live streaming for the same video are provided in two separate channels. For example, there is a channel for CCTV-6 which only provides live streaming service, while there is also another channel for CCTV-6 providing time-shifted service. Therefore, for UUSee, we measure the download rates of both the live streaming channel and the time-shifted streaming channel. In order to study the impact of the shifted time, we measure the download rate of the time-shifted streaming channel in two cases: one without any shifted time (referred to as No Delay), and the other with 20-minute shifted time (referred to as Delay).

Figure 1 shows that a peer in a DVR-style system (i.e., PPTV in the figure) is similar to a peer in a live streaming system (i.e. PPStream and UUSee Live) which has nearly a constant download rate. In contrast, the download rate of a peer in an NPR-style system (i.e., UUSEE Time-shifted) may be much higher than that in the corresponding live streaming system (i.e., UUSEE Live). It implies that an NPR-style peer fetches more segments than what it is viewing. The download rates in both UUSee Time-shifted (Delay) and UUSee Time-shifted (No Delay) are very close to each other, and gradually get close to the download rate of UUSee Live.

We claim that two independent channels in UUSee are not necessary. Actually we can use a single channel providing both live and time-shifted streaming. Furthermore using two separate channels prevents peers in different channels from helping each other, and may cause more segment requests to the server as demonstrated in our simulation.

III. NPR-style Time-shifted Streaming

Which segments peers prefetch is critical in NPR-style time-shifted streaming, and a good prefetching strategy results in high quality services and low server costs. Let us consider the case in Figure 2: peer i and j are requesting segment 6 and peer l is requesting segment 3. Peer k has upload capacity of 2 units. On the left side of Figure 2, peer k prefetches segment 3, then forwards it to peer l. The other two peers cannot get segment 6 from other peers, and directly request it from the server. In contrast, if peer k prefetches segment 6 as shown on the right side, it can fully utilize its upload bandwidth to forward segment 6 to both peers i and j. Peer l may be able to fetch segment 3 from other peers. Even if peer l fetches segment 3 from the server as shown in the figure, the server bandwidth consumption is still reduced by 1 unit compared to the left side of Figure 2.

A. System Model

A time-shifted system consists of a server S and $|N(t)|$ peers with p_i representing the i-th peer, U_i and D_i representing its upload and download bandwidth, respectively. The system allows peers to view past video with a maximum shifted time of T. The server continuously generates and injects new

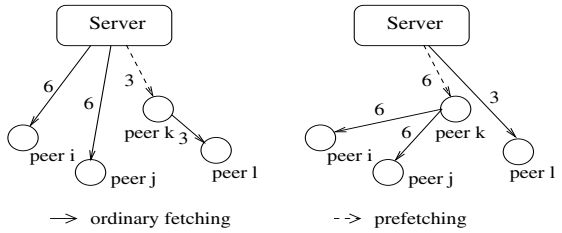

Fig. 2: In NPR-style streaming, a good prefetching strategy reduces server load without impairing video quality.

segments into the system, and the available segments are within the time-shifted window of $[t-T,t]$ where t is the current time. Let t_i denote p_i's viewing time at t, and we have $t-T \leq t_i \leq t$. Obviously, we can consider a live streaming system as a special case with $T=0$.

We assume that peers have sufficient storage to store all segments in the time-shifted window. As the window is usually of a limited size covering hours of the video, the assumption is reasonable. For example, a program such as a sitcom episode or a sports game lasts at most hours, and the time-shifted window covers hours of the video. Peers joining the system when the program is being broadcasted are able to view any part of the program from the beginning to the current time. The $[t-T,t]$ window moves forward along with t, and we use $M(t)$ to denote the set of segments provided by the system at time t. For example, in a system with a time-shifted window of 1 hour long, peers are able to view any video segments broadcasted between 7 pm and 8 pm at 8 pm. They can view any video segments broadcasted between 8 pm and 9 pm at 9 pm.

Let us consider peers' asynchronous viewing activities and their local memory status in the time-shifted streaming. Any segments within the $[t-T,t]$ window are available at the server, but not all of them are necessarily possessed by a peer. We use $I_i^j(t)$ to denote if p_i has segment j at t. Let a 0-1 variable $V_i^j(t)$ represent whether p_i immediately views segment j at t. Peers can view any segments within $[t-T,t]$, which is determined by their subjective choices. They fetch segments according to these $V_i^j(t)$'s and their local memory status. Namely, they must fetch the segments which they will view, but are not yet in their local memory. Peers also prefetch some segments to improve segment availability at peers, and we use $P_i^j(t)$ to show if p_i prefetches segment j at t. The prefetched segments may be viewed in the future or just be downloaded for forwarding to other peers. The value of $P_i^j(t)$ does not affect peers' current viewing experience as the segment is not emergent, but it affects other peers or the performance in the following time slots. $V_i^j(t)$ and $I_i^j(t)$ are known in our context, and $P_i^j(t)$ is the variable that is optimized based on the values of $V_i^j(t)$ and $I_i^j(t)$.

B. Problem Formulation

Our primary objective in this paper is to design an efficient prefetching method in peer-assisted time-shifted streaming, so that peers are able to get most segments before their playback deadlines, and most of the segments are fetched from other peers instead of the streaming server.

In a time-shifted streaming system, peers are able to view any segments within the window $[t-T,t]$. The segment requests which could not be served by other peers are finally sent to the server. Segment prefetching improves the supply of the segments, but on the other hand, segment prefetching also consumes bandwidth and it should not affect users' viewing experience. Then the problem can be formulated as:

$$min \sum_{j\in M(t)} \Big(\sum_{i\in N(t)} V_i^j(t+1) * (1-I_i^j(t+1)) - \sum_{i\in N(t)} I_i^j(t+1)U_i \Big)$$

$$s.t. \sum_{i\in N(t)} (V_i^j(t) * (1-I_i^j(t)) + P_i^j(t)) \leq \sum_{i\in N(t)} I_i^j(t)U_i,\ \forall j \quad (1)$$

$$\sum_{j\in M(t)} \sum_{i\in N(t)} (V_i^j(t) * (1-I_i^j(t)) + P_i^j(t)) \leq \sum_{j\in M(t)} \sum_{i\in N(t)} I_i^j(t)U_i,\ \forall i,j \quad (2)$$

$$\sum_{j\in M(t)} (V_i^j(t) * (1-I_i^j(t)) + P_i^j(t)) \leq D_i,\ \forall i \quad (3)$$

$$I_i^j(t+1) = I_i^j(t) + V_i^j(t) * (1-I_i^j(t)) + P_i^j(t) \quad (4)$$

$$P_i^j(t) = 0,\ if\ I_i^j(t) = 1 \quad (5)$$

In this formulation, $P_i^j(t)$'s determine the prefetching solution. They are binary optimization variables. The objective is to find the optimal prefetching variables $P_i^j(t)$'s to minimize the requests sent to the server.

Inequalities (1) and (2) are constraints ensuring that prefetching should not affect peer viewing experience. Inequality (3) is peer download bandwidth constraint. Eq. (4) is the updating equation for peer's local status. Eq. (5) is used to remove the duplicated fetches. Both the prefetched segments and the regular fetched segments are stored in peers' local storage and peers can forward them to other peers in the following time slots.

According to the objective formula, we have two ways to minimize the number of outstanding requests: decreasing the requests and increasing the supply. What peers are going to view is completely determined by themselves, the optimization objective can be expressed as a maximizing problem after expanding the minimization formula:

$$max \sum_{j\in M(t)} \Big(\sum_{i\in N(t)} V_i^j(t+1) * I_i^j(t+1) + \sum_{i\in N(t)} I_i^j(t+1)U_i \Big) \quad (6)$$

We update Formula (6) using Eq. (4). As peers' local status ($I_i^j(t)$), what peers are viewing ($V_i^j(t)$) and what they will

view in the next time slot ($V_i^j(t+1)$) are known in our context, the terms which consist of these variables are fixed. We have the equivalent problem after updating the optimization problem by removing these terms:

$$max \sum_{j \in M(t)} \Big(\sum_{i \in N(t)} V_i^j(t+1) * P_i^j(t) + \sum_{i \in N(t)} P_i^j(t) U_i \Big) \tag{7}$$

Based on Formula (7), we know that prefetching is affected by what peers are going to view and their upload capacities.

C. Heuristic Distributed Algorithm

The whole system information is required to calculate the optimal prefetching solution. For example, we need to know what peers are viewing, which segments exist in their local storage, how much their available upload bandwidth is, and so on. Though the optimal prefetching gives the lower bound of the server bandwidth consumption, the objective in this paper is to design a practical solution that can be implemented in a decentralized way.

We propose our heuristic distributed algorithm by decomposing the global optimization problem into multiple small optimization problems running on individual peers. For each single peer, it collects the local information from its neighbors, and then determines which segments to prefetch. Similar to the global optimization problem, we have the local optimization objective: a peer prefetchs segments so that it can forward the segments required by their neighbor as many times as possible. Thus, we have the localized format of Formula (7):

$$max \sum_{j \in M(t)} \Big(\sum_{k \in NBR_i(t)} V_k^j(t+1) * P_i^j(t) + P_i^j(t) U_i^j(t) \Big) \tag{8}$$

Where $NBR_i(t)$ is the neighbor set of peer i, and $U_i^j(t)$ is peer i's upload bandwidth allocated for segment j.

Similarly, the constraints also have the corresponding local format by replacing N_t with $NBR_i(t)$ and U_i with the bandwidth allocated by peer i.

Usually, it is hard to know the accurate bandwidth that a peer allocates to a specific segment in a distributed system. Further, a peer is both a request sender and a request receiver. Therefore we divide the algorithm into the request generation sub-algorithm and the bandwidth allocation sub-algorithm. A peer uses the request generation sub-algorithm to figure out the most demanding segments of its neighbor peers, and sends prefetching requests for these segments first. It uses the bandwidth allocation sub-algorithm to determine which received requests should be served, to increase the supply of the most demanding segments. Both of them have the same objective in Formula (8). In the request generation sub-algorithm, a peer does not just find the most demanding segments, but also determines how much bandwidth it can allocate for these segments. The bandwidth is used to forward the segments to other peers after the peer receives them.

In our simulation, peer i periodically calculates its residual bandwidth using its upload bandwidth to subtract the consumed bandwidth in the previous period. It determines which segments to prefetch based on the demand and supply of the segments from its neighbors. Then it calculates bandwidth that it can allocate to these requested segments, which is called the committed bandwidth. The segment requests are sent with this committed bandwidth information. If the peer does not have enough residual bandwidth to compensate the bandwidth that it consumes to prefetch a segment, it stops prefetching. When peers receive the requests, they always serve the requests which are for the rarest segments and/or the ones with higher committed bandwidth first. Tables (II) and (III) are the pseudo code for request generation and bandwidth allocation sub-algorithm on a single peer, respectively.

```
if p_i is going to view segment j in the next time slot then
  Checking if it is in the local memory I_i^j = 1
  if not in local memory then
    p_i requests segment j
  end if
end if
for any other segments within the shifted window do
  if p_i does not have them then
    p_i checks its neighbors' memory status to find the rarest
    segment
    calculates its residual bandwidth
    if residual bandwidth > bandwidth consumed by download-
    ing a segment then
      requests it with committed bandwidth
      min(residual bandwidth, 2*bandwidth consumed by seg-
      ment prefetching)
    else
      break
    end if
  end if
end for
```

TABLE II: Pseudo code: which segments to request.

```
for all received requests do
  if it has residual bandwidth then
    serves the requests for segments which sending peers need
    to view in the next time slot
    if still has residual bandwidth then
      finds the requests for the rarest segment in its neighbor-
      hood
      serves the one with highest committed bandwidth
      if the committed bandwidth is equal, randomly choose
      one of the requests
    end if
  end if
end for
```

TABLE III: Pseudo code: which requests to serve.

We can see that no global knowledge or synchronization are required in this algorithm, and the prefetching solution of a peer does not depend on the solution of another peer. Thus, the algorithm is scalable and practical.

IV. Performance Evaluation

We developed a simulator to evaluate the performance of our method (referred to as NPR), comparing with the following four methods. (1) Initial Playback Position Caching (IPP)

proposed in [2]: peers cache the content starting from where they start viewing; (2) Live Stream Position Caching (LSP) proposed in [2]: peers cache the content starting from live streaming playback points when joining the system [2]. For example, a peer joins the system at 8 pm, and chooses to view the video content at 7 pm. It caches the content starting from 7 pm in IPP, while it caches the content starting from 8 pm in LSP. (3) UUSee style method (referred to as NPR2) which has two channels for the same program: one for live streaming and the other for time shifted streaming. Please note that NPR and NPR2 are the same when there is only live streaming or only time shifted streaming. NPR2 is used as a reference to show that whether we need to separate peers into two channels providing live streaming and time shifted streaming, respectively. (4) The optimal method (referred to as Optimal) which is a centralized algorithm solving Problem (1) is used as a reference method. There are some other related methods [6], but they are tree-based. Since we implement a mesh-based overlay which is used in most commercial systems, and applying these methods in a mesh-based overlay will make them quite different from the original ones. Therefore they are not included in the simulation.

In our simulation, there are 1000 peers by default, and they have heterogeneous bandwidth: 20% peers have the upload bandwidth of 3 segments per time slot and download bandwidth of 10 segments per time slot. 80% peers have the upload bandwidth of 1 segment per time slot and download bandwidth of 5 segments per time slot. In all experiments, the server constantly generates 1 segment per time slot, and a maximum of 300 past segments are available by default. The server directly serves 10 peers. When peers cannot fetch segments that they will view in the next time slot from other peers, they send their requests to the server. In order to focus on the impact of these methods on the server bandwidth cost, we assume that the server has unlimited upload bandwidth so that the server can serve all peer requests and all peers can smoothly view the video. We define the server bandwidth cost to be the percentage of the server bandwidth consumption in the whole system bandwidth consumption. For example, if 50% segments streamed to peers are provided by the server, the sever bandwidth cost is 50%.

To compare NPR2 with NPR, we simulate 50% live streaming peers and 50% time-shifted streaming peers. Intuitively, the more live streaming peers that exist in the system, the closer the system is to a live streaming system, and vice versa. As NPR2 is used to show whether the system should be separated into live streaming and time-shifted streaming, we set 50% peers to be live streaming peers to make the system neutral. We simulate two separated channels for each of them in NPR2. There is only a time-shifted channel in all other methods, and both the live streaming peers and time-shifted streaming peers join this channel. We sum up the total server bandwidth cost in the two channels in NPR2, and calculate the percentage in the whole system bandwidth cost.

We implement the following 4 sets of experiments to study 1) the impact of peer churn, 2) the impact of the maximum shifted time, 3) the impact of initial playback point distribution and 4) the impact of dynamic viewing activities.

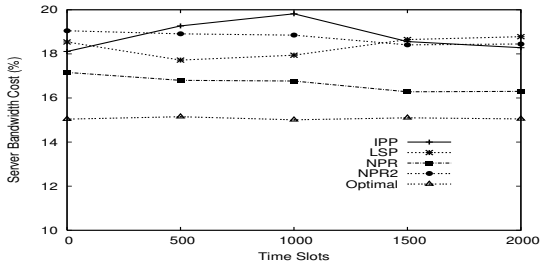

Fig. 3: NPR considerably reduces the server bandwidth cost through prefetching.

A. Impact of Peer Churn

In this group of simulations, we study whether these methods are robust to peer churn. Peers dynamically join the system at an exponentially distributed rate with a mean of 10 peers per time slot and their online durations follow an exponential distribution with a mean of 500 time slots. A peer randomly chooses a playback time between $t - T$ and t according to the uniform distribution. Figure 3 shows their performance: our NPR method consumes less server bandwidth than IPP, LSP and NPR2. Because in our method, peers utilize the spare bandwidth to fetch segments which are not required by themselves but highly demanded in the system, and this improves the segment availability at peers. NPR2 prevents peers in a channel to help peers in another channel, and thus its server bandwidth cost is higher than NPR. However, peers have limited neighbors and their decisions are based on local information in our simulation. There is a performance gap between our method and the optimal one. It is more obvious when the system changes more dynamically, as shown in a following section.

B. Impact of the Maximum Shifted Time

In this group of simulations, we investigate the impact of the maximum shifted time on the server bandwidth cost through changing the value of T from 150 to 600 time slots. As shown in Figure 4, the longer the maximum shifted time is, the higher the server bandwidth cost is for all the methods. It is easy to understand: peers have more playback points to choose with a longer maximum shifted time. Therefore, the segment sharing becomes more challenging. Since, peers are able to identify which segments are more scare using our prefetching method, the server cost increases more slowly with NPR than with IPP and LSP.

C. Impact of the Initial Playback Point Distribution

In a time-shifted streaming, peer viewing time may not be uniformly distributed between $t - T$ and t. For example, the initial playback points are more likely to be close to the beginning of a video than to the end of the video. We equally divided the video in $[t-T, t]$ into three sections, which represent the beginning, the middle and the end, respectively.

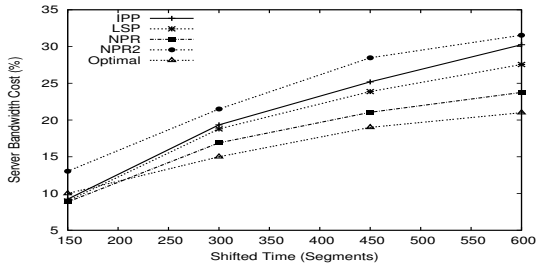

Fig. 4: The longer the maximum shifted time, the higher the bandwidth cost. Prefetching helps NPR alleviate the impact.

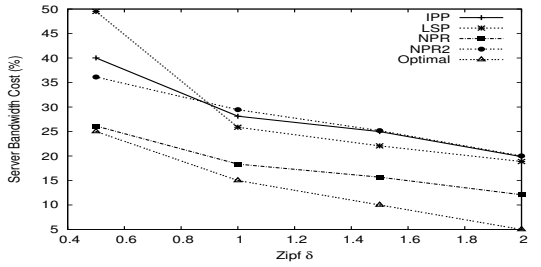

Fig. 5: NPR pretching method achieves low server bandwidth cost with different initial playback point distributions.

We conduct the experiment in which peers' initial playback points are in one of three sections. The numbers of peers in these three sections follow Zipf's law which is an empirical law formulated using mathematical statistics [1]. The Zipf parameter controls the number of peers viewing each section. Namely, if the Zipf parameter is small, most of peers are viewing the end section; whereas most of peers are viewing the beginning section when the parameter is large. A peer firstly determines which section it wants to view, then it randomly chooses the viewing time in the corresponding section. For example, if a peer chooses to view the beginning section, its viewing time can be any value in $[t - T, t - \frac{2T}{3}]$. Figure 5 indicates that the more peers view the beginning section, the less the server bandwidth cost is. Because usually the segments in the beginning section are cached more than the recent segments. In our NPR method, peers determine which segments to fetch and cache based on the segment availability and demand, so it always consumes less server bandwidth.

D. Impact of Dynamic Viewing Activities

We also show the impact of peer jumping backward and forward within the shifted time window. Peers jump to view other video segments after they join the system. The intervals between a peer's two consecutive jumps are uniformly distributed between 0-500 time slots. In Figure 6, we can see that the server bandwidth cost is very high for all the methods. But our method steadily decreases the server bandwidth cost after 500-600 time slots. As our NPR method forces peers to aggressively fetch segments using peers' spare bandwidth, they get a considerable number of segments in its local memory. Thus peers are able to view the segments from their local memory or fetch them from other peers.

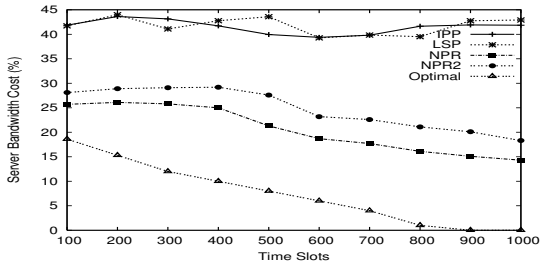

Fig. 6: Peers randomly jump backward/forward, average 10 times per peer in this experiment. NPR prefetching method reduces the server cost over time.

V. Conclusion

In this paper, we studied the design of NPR-style time-shifted streaming with low server cost using segment prefetching. We formulated it as an optimization problem with the objective of minimizing the server bandwidth cost. Peers' idle bandwidth is utilized efficiently, so that all peers receive smooth streams. Then we proposed our distributed heuristic algorithm called NPR method, with which each peer determines how to send segment prefetching requests and how to allocate its bandwidth. The simulation results show that our NPR method has obvious improvement with respect to the server bandwidth costs. On the other hand, prefetching in our method consumes extra peers' bandwidth, and the peers might not view the prefetched segments themselves. We are very interested in measuring this bandwidth consumption, and will study this in the future.

References

[1] http://en.wikipedia.org/wiki/Zipf's_law.
[2] S. Deshpande and J. Noh. P2TSS: Time-shifted and live streaming of video in peer-to-peer systems. In *Proceedings of IEEE Multimedia and Expo*, Hannover, June 2008.
[3] M. Guo and M. H. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.
[4] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller. Liveshift: Peer-to-peer live streaming with distributed time-shifting. In *Proceedings of IEEE P2P*, Maastricht, Netherlands, September 2008.
[5] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.
[6] J. Noh, A. Mavlankar, P. Baccichet, and B. Girod. Time-shifted streaming in a peer-to-peer video multicast system. In *Proceedings of IEEE Globecom*, Hawaii, November 2009.
[7] Time Shifting. http://en.wikipedia.org/wiki/Time_shifting.
[8] UUSee. http://www.uusee.com.
[9] M. Wang, Lisong Xu, and Byrav Ramamurthy. A flexible divide-and-conquer protocol for multi-view peer-to-peer live streaming. In *Proceedings of IEEE P2P*, Seattle, WA, September 2009.
[10] M. Wang, Lisong Xu, and Byrav Ramamurthy. Linear programming models for multi-channel p2p streaming systems. In *Proceedings of IEEE INFOCOM - mini conference*, San Diego, CA, March 2010.
[11] Wiki. http://en.wikipedia.org/wiki/Time_shifting.
[12] 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*, San Francisco, California, November 2006.