Understanding the impact of VCR operations in P2P VoD systems

Xiaoyuan Yang, Pablo Rodriguez
Telefonica Research
{yxiao,pablorr}@tid.es

Minas Gjoka, Athina Markopoulou
University of California, Irvine
{mgjoka, athina}@uci.edu

***Abstract*—One of the main challenges faced by peer-to-peer (P2P) video-on-demand (VoD) systems is their ability, or lack thereof, to provide VCR functionality (e.g. random jump operations). Such jump operations can have special impact in the server load during flash crowd events, since the server is the only node in the system with a full copy of the video. Currently, there is little knowledge of the impact that such VCR operations can have in the swarm efficiency (and therefore in the playout rate and start-up delay), as well as in the server's load under different jumping and arrival patterns. In fact, existing state-of-the-art approaches rely mostly on highly over-provisioned peers and servers to absorb such jumping operations.**

In this paper, we provide the first evaluation of the impact that VCR operations can have on P2P VoD systems for various arrival and viewing patterns. We provide both analytical evidence and experimental results. To better evaluate the impact of these patterns, we have built a system called Kangaroo, which provides advanced DVD-like features, namely arbitrary forward/backward jumps, using an entirely mesh-based network. The system achieves its goals through a combination of carefully designed mechanisms, including proper block scheduling algorithms and peer-neighborhood topology management. We also study how viewing patterns affect the performance of the system and use traces captured from a live VoD system, to characterize the performance of the system under realistic workloads.

I. Introduction

Peer-to-peer (P2P) systems have been successful in distributing files to a large number of users. P2P systems are also widely used for distribution of video content, including video downloads (where users need to completely download the file before they can watch the video) and live media streaming (such as Coolstreaming). Recently, new systems [1] have been designed to enable a VoD experience using P2P. However, in such video-on-demand (VoD) services, assume that viewers watch from start to finish at the playback rate. Supporting VCR functionality (e.g. pause/resume, backward and forward jumps across the video) is a natural requirement for most of the VoD systems. Although the most popular centralized VoD services such as YouTube or home theater solutions support such seeking functionalities, VCR functions have been ignored by many P2P VoD systems. Understanding of the performance impact of VCR operations and optimizing the system design for their support is required step towards fully functional P2P VoD systems

Designing a VoD system with advanced features using P2P technologies is non-trivial because of the lack of synchronization among users, which reduces the P2P sharing opportunities. This is especially critical during flash crowds, since the server is the only node in the system with a copy of the file. As users jump around the video sequence, the chances for sharing decrease, and receiving the video in a timely manner often requires pulling data from the server. The difficulty lies in designing a P2P system which meets the VoD requirements (low delay from a user's perspective), high utilization of the system resources (system/provider's perspective), and minimizing the amount of data that has to be delivered from the server under arbitrary jump patterns. In the ideal scenario, the server should only be used if there are enough peers in the system that can satisfy the demand created by VCR functions. However, finding the right peers on time or freeing their resources is non-trivial. In fact, current state-of-the-art approaches for VoD over P2P (even without VCR functionalities) rely either on greatly over-provisioned servers or require the users to wait for long start-up delays.

Interestingly, regardless of how well one designs a P2P VoD protocol to support VCR operations, for certain jumping patterns the load at the server is inevitable. For instance, a server may require $O(n)$ capacity if n users request simultaneously to view a different segment in the same video. Therefore, the server load heavily depends not only on the system design but also on the arrival pattern and viewing profile. Still, little is known about how such parameters affect the server requirements.

In this paper, we provide the first study of the impact that jump operations can have in the server load and in the P2P VoD experience. To this extent, we characterize the jumping behavior of real users using traces collected from a large operational IP-TV network and our own corporate P2P VoD system and provide both analytical as well as experimental results using an array of jumping and arrival patterns, including a model based on our real-world traces.

To experimentally validate our results, we have built and deployed with real users *Kangaroo* - a P2P mesh-based system that can efficiently deal with "jumps", i.e. provide users with a high-quality VCR service while requiring only a small capacity increase at the server. Mesh-based P2P has been successfully used for bulk file dissemination but it has been so far unclear whether they can successfully achieve the sequential segment delivery needed for continuous playout in the presence of arbitrary jumps. Kangaroo achieves these goals by making careful design choices in several aspects of the system. In particular, it implements a scheduling policy that combines selfish with altruistic behavior for continuous playback and to improve block diversity. It also implements a topology manager that helps peers at similar playback points to mesh with each other and to quickly find peers with the desired data segments during jump operations and a scalable tracker that quicks in when needed and quickly supplies new useful peers.

The results obtained for both the ideal P2P VoD system and for Kangaroo indicate that VCR operations have a significant impact on the server bandwidth consumption which can be, however, reduced if the system is properly optimized for VCR operations. [PABLO: put some numbers for low start up delay, scalability of tracker, etc...]

II. Related Work

Live video streaming p2p systems were among the first p2p video systems considered. Those included tree-based overlays such as Slipstream and mesh-based overlays such as Coolstreaming, PPLive. Next generation video p2p systems have been designed to support VoD (e.g. BitOS, Redcarpet[1], Toast). More recently, [6] and [4] considered a P2P system that can support VCR-like functionalities. In particular, Gridcast ([4]) introduces the concept of anchors to prefetch data in predefined points of the video and allow for random jumps. Bulletmedia ([6]) proposes a more aggressive proactive caching which creates multiple copies of every segment. However, both systems focus on efficient distributed algorithms and data structures to help peers find the desired data. In contrast to these works, we have taken a more holistic approach to understanding VCR like operations, trying to determine what are the fundamental trade offs and limitations that are innate to a particular arrival and jumping patterns and what is the impact in the server load and user experience. We have also implemented a system that does not rely on aggresive data prefetching and that relies on a simple tracker to ensure that peers find the required data in a timely fashion and with minimum server load.

III. The Impact of the Workload

In this section, we study the impact of different workloads (jumping patterns) on the capacity of the source.

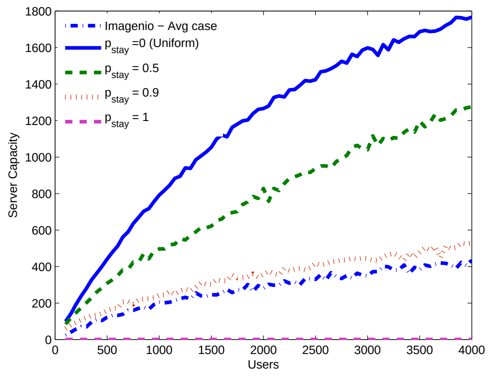


Fig. 1. Source capacity C_{max} in an ideal system (BestP2P) for workloads with various degrees of continuity. The file has 2048 segments.

First, we consider an ideal system, called *BestP2P*, which allows us to focus on the workloads themselves, while abstracting away implementation-specific bottlenecks. In later sections, we also design and evaluate a real system, *Kangaroo*, to experimentally evaluate the workloads.

In BestP2P, peers have infinite upload bandwidth and number of active upload connections which allows them to disseminate instantaneously all segments, once provided by the source peer. In each timeslot, peers request different segments from the source. The source must have enough capacity to serve immediately all requests for segments that have not been requested before; but it relies on the peer-to-peer swarm to distribute already served segments. Therefore, the demand at the server at time t, $D(t)$, is the number of newly requested segments. To provision for the maximum demand, the source needs capacity $C_{max} = \max_t D(t)$. Alternatively, a provider might prefer to provision for C_{95} - the 95^{th} percentile of demand $D(t)$.

We are interested in understanding how the workload affects the capacity C_{max}at the server. The problem can be formulated as a balls-in-bins problem: in each time slot n balls (one request from each peer) arrive to m bins (segments of video) according to the workload (jumping pattern). The problem is to characterize demand at the server $D(t)$ and the capacity required to accommodate all the demand C_{max}. Let us discuss some interesting workload cases.

Deterministic workloads. In the *worst case*, each peer requests a different new segment in the same timeslot, thus leading to capacity $min\{m, n\}$. As time goes by, there is a decreasing number of segments that have not been requested before and the demand at the source is decreasing. In the *best case*, the capacity is 1: at most one new segment is requested in the same timeslot; e.g. this happens when users watch sequentially.

Random workloads. The worst random jumping pattern is *uniform*: in any time slot, a peer can request any randomly chosen segment. We can show (proof omitted for lack of space) that the capacity required is on average

$$E[C_{max}] = m(1-(1-\frac{1}{m})^n) \simeq m(1-e^{-\frac{n}{m}}) \qquad (1)$$

This describes the scalability of the system with the number of segments m and the number of users n. Simulation results, shown in solid blue line in Fig.1, perfectly agree with the above formula (plot omitted). In reality, users do not randomly jump to any segment at any time but instead alternate between *continuous playout* and jumps. Fortunately, continuous playout patterns make the provisioning problem easier. Consider, for example, that each peer continues to play the next segment with probability p_{stay} or switches to a random segment with prob. $1-p_{stay}$. The results are shown in Fig.1: the more continuous the playout (large p_{stay}), the smaller capacity is required by the server. At one extreme, $p_{stay}=1$ is the best case of sequential playout that requires $C_{max}=1$; at the other extreme, $p_{stay}=0$ corresponds to the uniform pattern, which requires the highest capacity.

Anchor points. In the case where random jumps are restricted only to a selected set of *anchor points*, we found that the capacity of these systems scales with the number of users better than their non-anchored counter parts. The exact formula for C_{max} depends on the number of anchor points relative to the total number of segments. E.g. the workload for $p=0.9$ behaves similarly for a large (more than 500) number of anchor points (plot currently omitted from Fig.1); but it behaves much better for a small number of anchor points; e.g. $C_{max} \leq 100$ for 100 anchors and $C_{max} \leq 170$ for 200 anchors.

Realistic Workload (Imagenio Traces). The question then becomes, how do workloads look in reality? Are they close to the worst or the best case? To answer this question, we studied VoD traces from Imagenio, an IPTV service over ADSL provided by Telefonica at Spain.

The Imagenio traces span over a period of 109 days with a total of 65,598 sessions. Each session corresponds to a user watching a certain movie and contains a number of events triggered by the user. A movie title may be present in more than one session. In this paper, we focus on sessions longer than 10 minutes, because these are the ones exhibiting interesting jumping patterns. A user in the Imagenio traces can be in one of four states: Play, Pause, Fast Forward and Fast Backward, with probability 0.54, 0.09, 0.31 and 0.04 respectively. We can translate the time skipped when the user plays faster than the normal mode to the corresponding jump distances that happen instantaneously. We looked at the time a user stays in Play and Pause mode as well as the jumping distance for the Forward and Backward jumps; all four empirical distributions can be fitted by a lognormal distributions with different parameters that can be estimated from the data. Details on the model are omitted for lack of space.

From the probabilities and durations of the events, one can see that there is high degree of continuity in the Imagenio traces. This means that this is a benign workload that allows for good performance without over-provisioning the system. Indeed, when we simulated the Imagenio jumps for the ideal BestP2P system, we obtained the dash-dotted line shown in Fig.1, which requires small capacity. Experimental results in section V show that this is also the case when Imagenio traces are used as input for the real Kangaroo system.

IV. System Design

In order to evaluate the effect of jump operations via experiments on a real system, we also built a mesh-based p2p system, called *Kangaroo*. In this section, we describe some of the system mechanisms that are key to providing the functionality of interest, i.e. jump operations for VoD.

A. Overview

There is one swarm per media file. A media file is split into segments of equal size. The peer containing the entire media file is referred to as the *source* peer. Each peer maintains control connections with 10-15 neighbor peers and keeps a list of 10 more additional collaborator peers which serve as potential neighbors. Each peer can have up to 5 active upload and download data connections with its neighbors at any moment. Data connections are rate-limited. We use persistent HTTP/1.1. connections to allow for seamless integration with web proxies.

B. Topology Management

A key design choice was the use of a centralized tracker for discovering and meshing together peers with similar playback times. A peer contacts the tracker after joining as well as after a Jump event to report its playback position. The tracker uses a simple distance ranking, based on the projected playback position of every peer since its last contact, to match peers with similar playing positions. For scalability, minimum state at the tracker is required.

Each peer has a topology manager module which reshuffles the peer's neighborhood when necessary. Every 5sec, a peer calculates the health factor h of its neighborhood as the average health of individual neighbors. The health of a neighbor is the percentage of useful segments this neighbor has; useful segments are considered those within a window from the currently playing segment. If h is less than a threshold t then (i) the neighbor relations with the non-useful peers are reset (ii) more collaborators are requested from the tracker and (iii) new neighbor relationships are created. A gossip mechanism is also used: when a peer refuses to upload a segment it also suggests the location of another peer that has the segment.

C. Segment Scheduling

Another key mechanism for providing continuous playout in the presence of jumps is the scheduling policy used by the peer to upload and download segments.

The *download scheduler* faces the following tradeoff: on one hand the peer wants to download the next segments for its own continuous playback; on the other hand it wants to download the local-rarest segment to help the swarm performance. We chose to use a hybrid sequential and local-rarest policy. The next segment to download is chosen randomly among a list of five candidate segments consisting of: the next four segments after the currently playing and one local rarest segment. Once the segment is chosen, the neighbor from which to download is chosen as follows: (i) the neighbor with the least number of segments so as to reduce the chances of downloading content from the source; (ii) the neighbor with the least number of active data connections, so as to avoid rejections at the upload scheduler. Furthermore, the hybrid policy varies the ratio of local-rarest vs. continuous segments in the list, depending on the segments are available at the peer's buffer for continuous playback; thus, fast peers can contribute to the swarm without affecting their own needs.

The *upload scheduler* of the source accepts to serve worst seeded segments and can reject to serve other requests; the rational is to avoid sending duplicates and thus waste resources. There is one exception to this rule: a request is always accepted if the segment has exceeded a delay-tolerance parameter, which captures the maximum delay that a peer can tolerate for each segment. If the delay tolerance is exceeded and the peer cannot find the segment from the swarm, then the peer can go to the source as a last resort; this policy bounds the jump delay at the expense of potentially more capacity at the source.

V. Experiments

In this section, we apply different workloads as input to the Kangaroo system and study how these affect the user performance and system provisioning.

A. Experimental Setup

We implemented a multi-threaded prototype of Kangaroo. The peers collaborate for the duration of their playback and for 10sec afterwards. The upload and download rate of each peer are the same, α, a multiple of the playback rate; this is a tunable parameter of the system desired toe be close to 1. There are 172 peers in all experiments, unless otherwise mentioned. Each peer follows its own schedule of jump operations and can jump at any time without any initial system stabilization period. We show results for a video file with duration 1024 seconds, split into 2048 segments of 64KB each, and playback rate of 1Mbps. The experiments ran on a local LAN connected with a 100Mbps Ethernet switch.

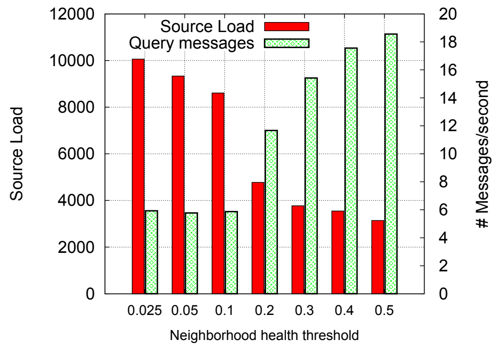


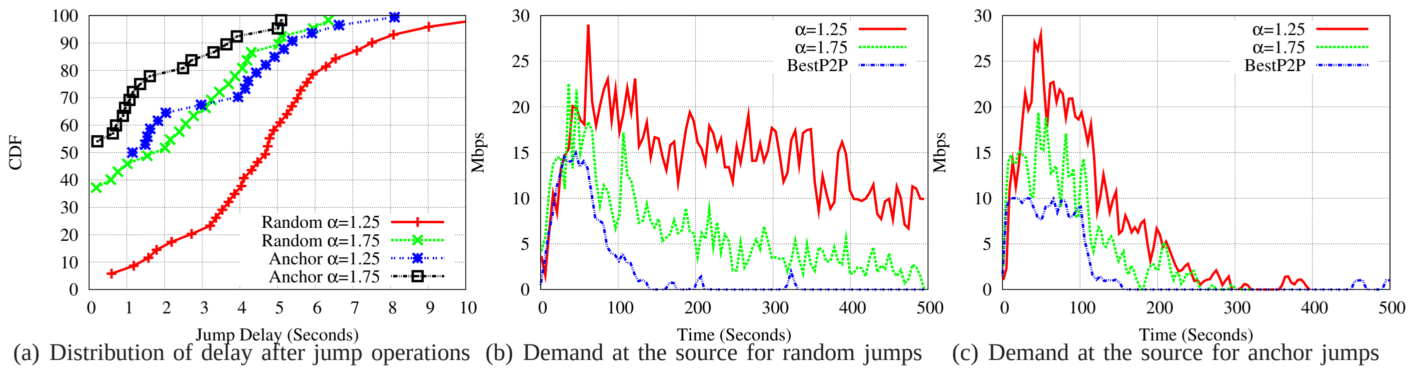
Fig. 2. Tuning the threshold for the neighborhood Health Factor.

We are interested in the performance of the system for different workloads. There are two characteristics of a workload that affect the performance: the arrival of users and their jumping operations. We have considered the following user arrivals types: (i) a flash crowd with all peers joining at the same time (ii) groups of peers joining in a batch at different times and (iii) Poisson arrivals. We have also considered the following jumping patterns: (i) sequential playback (ii) random jumps at any points of the video (iii) random anchor jumps, where jumps are allowed only to a set of anchor points and (iv) Imagenio traces. We performed experiments for all the combinations of these arrival and jumping patterns; below we showcase the results for some interesting cases.

We are interested in the following performance metrics. From a user's point of view, a small delay is desired after each jump operation, thus termed *jump delay*. From the system's point of view, the capacity/bandwidth at the source (α_s) and at the peers (α) should be kept low; the tracker should also scale with the number of users. A well-designed system should provide the users with good experience without over-provisioning.

B. Tracker Load

The number of requests the tracker receives is important for the scalability of the system. Peers contact the tracker to update their neighborhood in two cases: (i) at every jump operation or (ii) when triggered by neighborhood health evaluation. The former depends on the workload while the latter depends on the value of the threshold t of acceptable health factor. The choice of t involves a tradeoff: higher values of t lead to better neighborhoods but also increase the number of messages to the tracker, which can increase the response time of the tracker and eventually the delay experienced by peers at each jump. The threshold t also indirectly affects the load of the source peer, since healthy neighborhoods increase the efficiency of the swarm and reduce the number of requests sent to the source. Fig.2 shows the number of triggered updates and the source load as functions of the threshold



(a) Distribution of delay after jump operations (b) Demand at the source for random jumps (c) Demand at the source for anchor jumps

Fig. 3. Kangaroo's performance for Flash Crowd arrivals with Random Jumps.

TABLE I
NUMBER OF SEGMENTS PROVIDED BY THE SOURCE FOR FLASH CROWD AND RANDOM JUMPS

	Client/Server	Multicast	BestP2P	K: α=1.25	K:α=1.75
Random Jumps	192.229	116.658	2.259	15.728	6.337
Anchor Jumps	83.806	60.755	2.114	5.970	3.840

t, for 172 peers arriving in a flash crowd and doing exactly two random jumps forward. In this experiment, a value of $t = 0.2$ gives a good trade-off between the tracker response time and topology connectivity; the average request load at the tracker by each peer is 34.7bps. Therefore, a tracker with 10Mbps upload rate can potentially support up to 179,793 peers. More generally, t should be chosen so as to combine a low number of tracker updates and high performance.

C. Sequential Playback

We started by looking at the simplest viewing pattern, where peers play sequentially from the beginning until the end. However, we combine it with two challenging arrival patterns. The first is Flash Crowd: 172 peers join the swarm at the same time. The second is a Batched Join scenario: 7 groups of peers (25 peers per group) join the system every 30 seconds. We fixed the upload rate at the source at 2Mbps ($\alpha_s = 2$). Experimental results showed that only 4 seconds of setup delay are required for $\alpha = 1.25$ in the flash crowd and $\alpha = 1.5$ in the batched join scenario. The latter is more challenging because peers joining later delay the stabilization of data paths in the swarm. A measure of how well the system is doing is the variance of download time for peers in the same group; we found this to be less than 1% for all groups.

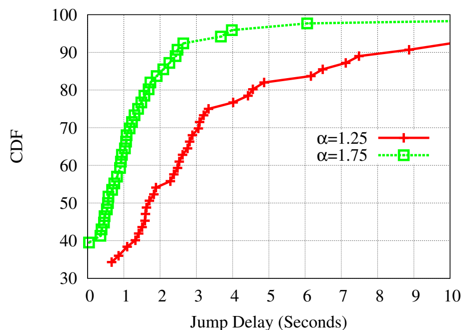
We also studied a heterogeneous scenario with asymmetric upload/download capacities, different among peers, as expected in the real Internet. For this environment, we modified the scheduler for setting the number of upload/download connections proportional to the upload/download bandwidth. We found that the playback rate in this setting is only slightly reduced, by less than 7%, compared to a similar homogeneous setting.

Another question is how the upload rate at the source α_s affects the system's performance. We fixed $\alpha = 1.75$ for all peers and vary only α_s between $2.0 - 4.0$: increasing α_s beyond 2.5 resulted in insignificant improvement. Although increasing the upload rate of the source reduces the system latency, the average system performance is mostly affected by the total upload bandwidth available by the peer-to-peer swarm. We also performed experiments with up to 300 peers with negligible performance deterioration.

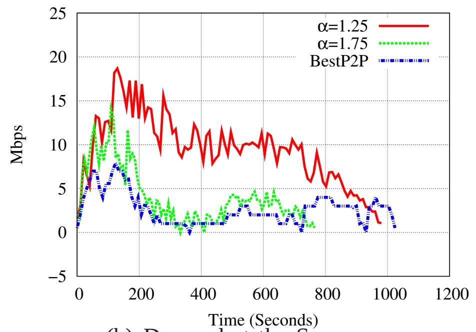
D. Random Jumps

We now study the impact of random jump operations on the system resource requirements. Similarly to our discussion in section III, consider that each peer can switch from sequential playing to jumping mode according to a fixed probability in each time slot. The jump distance is calculated according to a distribution and determines the new playback point after the jump. We did experiments with several distributions, and here we present the results for some of them. In *random jumps* the new playback point is chosen uniformly at random from the current playing position until the end of the video. In *anchor jumps*, there is a limited number of preselected anchor points in the sequence, and the playback point is chosen uniformly at random among anchors from the current playing position until the end; we show results for experiments with 10 anchor points at equal distances. We chose these distributions because they stress the system more, e.g. compared to backwards jumps and other patterns. We fix α for all peers but leave α_s unrestricted and observe the demand at the source.

Fig.3 shows the results obtained for Flash Crowd arrivals combined with these jumping patterns. Fig. 3(a) shows that most jump operations have less than 10sec delay even for small α's. Furthermore, a small increase in the a of all peers, from 1.25 to 1.75, can significantly reduce the jump delay and thus improve user experience. Between the two patterns, anchor jumps require smaller jump delay. Fig. 3(b) and 3(c) shows the bandwidth demand at the source as a function of time; the capacity at the source is the maximum demand across time.



(a) Jump Delay



(b) Demand at the Source

Fig. 4. Imagenio workload with Poisson arrivals. (Experimental results using Kangaroo.)

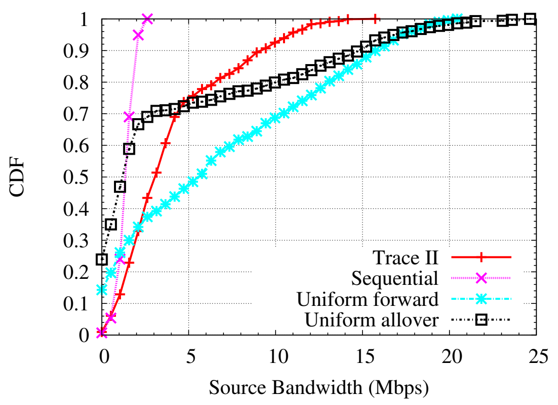


Fig. 5. Demand at the source for different workloads and Poisson user arrivals. (Experimental results using Kangaroo.)

Table I shows the total number of segments provided by the source during the entire experiment. Kangaroo can reduce the load at the source by 97% compared to client-server distribution, and by 95% compared to multicast-based distribution. Compared to the ideal BestP2P system, Kangaroo needs only slightly more than BestP2P for the same workload (2.8 times for random and 1.8 times for anchor jumps).

We also did experiments (omitted for lack of space) with the same jumping patterns and number of peers but Poisson arrivals distribution with $\lambda = 1$ peer/sec. Compared to a Flash Crowd, Poisson arrivals spread the demand for segments in time and thus require less system resources.

E. *Imagenio Traces*

We also did experiments using the Imagenio traces as the input workload to Kangaroo. We focused on the most popular video and normalized session lengths and jump distances to 1024sec, for a fair comparison to the previous scenarios. Each of the 172 users randomly chooses a session. Fig 4(b) shows the source upload rate required as a function of time. The total number of segments delivered by the source with $\alpha = \{1.25,1.5,1.75\}$ is $\{$17,442, 9,312, 6,155$\}$ respectively. In the optimal (BestP2P) system, the source delivers $5,409$ segments; therefore Kangaroo with $\alpha = 1.75$ sends only 1.13 more segments than BestP2P would for the same workload. Notice that Kangaroo performs better for the Imagenio than for the random jumps patterns, because the real traces contain several continuous playback and pauses. Fig.4(a) shows also good results in terms of delay: 95% of the jumps happen within 4sec, if peers use $\alpha = 1.75$.

Fig. 5 compares the demand seen at the source peer for different workloads and Poisson arrivals (with the same rate λ =1 peer/sec). In particular, we show the distribution of demand at the source, as experienced at different times throughout the duration of the swarm. Obviously, sequential playback is the easiest workload and requires $a_s \leq 2.5$. Random jumps at uniformly distributed points of the remaining sequence is the most demanding workload. Random jumps at uniformly distributed points in the entire sequence is an easier workload because some of the jumps are backward, which can be served by the swarm instead of the source. Finally, the Imagenio traces is a relatively benign workload: 75% of the time it requires $a_s \leq 5$ and 95% of the time it requires $a_s \leq 10$. The results of Fig. 5 can be used to provision the system, e.g. to accommodate 95% of the demand. These numbers are even better when compared to the capacity required by BestP2P for the respective workload (they were found to be close to that - details are omitted).

VI. SUMMARY

In this paper, we have studied the effect that jump operations can have on P2P VoD systems. In particular, we have considered different arrival patterns and viewing behaviors and how they fundamentally impact the performance and provisioning of a p2p system. To this extent, we have explored different such patterns analytically and shown experimentally that designing a system that provides a performance close to the best possible is feasible. In fact, our preliminary results using realistic jumping patterns and a real implementation show that one can achieve a good user experience without aggressively pre-fetching or over-provisioning the system.

References

[1] S. Annapureddy, S. Guha, C.Gkantsidis, D. Gunawardena, and P. Rodriguez, ”Is High-Quality VoD Feasible using P2P Swarming?”, *in Proc WWW 2007*

[2] S. Annapureddy, S. Guha, C.Gkantsids, D. Gunawardena,P. Rodriguez, ‘Exploring VoD in P2P Swarming Systems,” *in IEEE Infocom Mini-symposium 2007*

[3] B. Cheng, H. Jin, X. Liao, “Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays”, *in ICC 2007*.

[4] B. Cheng, X. Liu, Z. Zhang, H. Jin, “A Measurement Study of a Peer-to-Peer Video-on-Demand System ”, *in IPTPS 2007*.

[5] C. Huang, J. Li, K. Ross, “Peer-Assisted VoD: Making Internet Video Distribution Cheap”, *in IPTPS 2007*

[6] N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, A. Rowstron, “Enabling DVD-like features in P2P Video-on-Demand Systems”, *in ACM P2P-TV Workshop 2007*.

[7] Y. Choe, D. Schuff, J. Dyaberi, V. Pai, “Improving VoD Server Efciency with BitTorrent”, *in Proc of ACM Multimedia 2007*.