The Impact of Playout Policy on the Performance of P2P Live Streaming

. . . *or how not to kill your P2P advantage*

Constantinos Vassilakis[a], Nikolaos Laoutaris[b], and Ioannis Stavrakakis[a]

[a]Dept of Informatics and Telecommunications, University of Athens, Greece;
[b]School of Engineering and Applied Sciences, Harvard University, USA

ABSTRACT

In this paper we examine the impact of the adopted playout policy on the performance of P2P live streaming systems. We argue and demonstrate experimentally that (popular) playout policies which permit the divergence of the playout points of different nodes can deteriorate drastically the performance of P2P live streaming. Consequently, we argue in favor of keeping different playout points "near-in-time", even if this requires sacrificing (dropping) some late frames that could otherwise be rendered (assuming no strict bidirectional interactivity requirements are in place). Such nearly synchronized playout policies create "positive correlation" with respect to the available frames at different playout buffers. Therefore, they increase the number of upstream relay nodes from which a node can pull frames and thus boost the playout quality of both single-parent (tree) and multiple-parent (mesh) systems. On the contrary, diverging playout points reduce the number of upstream parents that can offer a gapless relay of the stream. This is clearly undesirable and should be avoided as it contradicts the fundamental philosophy of P2P systems which is to supplement an original service point with as many additional ones presented by the very own users of the service.

Keywords: Playout scheduling, video streaming, peer to peer streaming

1. INTRODUCTION

P2P streaming using tree or mesh overlays: Distributing a live video stream using a P2P streaming system has the advantage over a point-to-point client/server system of offering more resources to clients by effectively turning each one of them into a secondary server that assists in the distribution of the stream. These additional resources can yield improved scalability and/or resilience, depending on the design of the system. Organizing the nodes into a tree shoots for scalability by requiring only n overlay links, where n denotes the number of receivers. Having a minimal number of overlay links reduces the *stress* of the underlying physical links, i.e., the number of times that the same information can flow over the same physical link (this can happen as multiple overlay links may go through the same physical link). It also minimizes the amount of overlay link monitoring overhead for detecting congestion/churn and triggering handoffs in-time to avoid disruption of playout (assuming that the same amount of monitoring expenditure is paid at each link). These observations hold true to both single-tree/no-coding and multiple-tree/multiple-description-coding architectures.

Meshes on the other hand shoot for resilience to congestion/churn by providing each node with multiple parents from which it can receive the stream in parallel (using single-, multiple-description-, or network-coding). Since a mesh uses more than n overlay links, it can increase the stress of the underlying physical links and the monitoring overhead (although the latter is not necessarily true as a node can take advantage of the redundancy offered by having multiple parents and performing a more lazy monitoring of each incoming links). The discussion of which combination of topology and encoding is the "right-one" has been going-on for some time, and although it seems that recent mesh-based systems using coding have several advantages,[1] it all depends in the end on the assumed operating environment and the desired cost/complexity of building and maintaining the system: well-behaving environments (e.g., dedicated cable networks) can benefit from the simplicity/economy offered by tree-based distribution; uncontrolled/variable environments (e.g., under-provisioned parts of the current Internet) can benefit from the redundancy offered by mesh topologies and coding.

The (new) role of playout scheduling in P2P: The playout scheduler is the component of a video receiver that handles the buffering and rendering of received frames. Designing appropriate playout schedulers for video streaming application was one of the central research topics of the multimedia transmission community up to the emergence of P2P streaming systems, at which point the focus shifted onto overlay construction and coding issues. Although fairly well understood

Email addresses: cvassilakis@noc.uoa.gr (C. Vassilakis), nlaout@eecs.harvard.edu (N. Laoutaris), ioannis@di.uoa.gr (I. Stavrakakis).

in the context of point-to-point video streaming,[2] playout scheduling has received a rather limited attention in the context of P2P video streaming. The new setting, however, perplexes playout scheduling beyond our previous understanding. In addition to achieving the desired tradeoff between interactivity and stream continuity, the playout scheduler must now jointly factor-in that different playout processes become coupled in the context of P2P: buffering, rendering, or dropping a frame affects not only the local process but also downstream ones that might connect and request frames from the local node. In this article we argue that although seemingly subtle compared to topology construction and coding, playout scheduling still deserves some attention as a bad choice with respect to it can impact quite negatively the performance of P2P streaming systems, despite the existence of the other two powerful enablers.

Our contribution: We consider a *delay preserving* playout policy called *Sync* and a *data preserving* one called *Async* in the context of a P2P streaming system. These two policies lay at the extremes of the spectrum of studied playout policies.[2] With Sync, the playout scheduler enforces a fixed predefined time offset between the time that a frame is presented at a receiver and the time it was captured at the sender. To do so, it has to drop "late" frames that arrive after their scheduled playout time, even if they are eventually received correctly and in their entirety. The data preserving Async policy on the other hand, imposes an initial buffering delay and then presents frames by draining the buffer at a constant rate. In the event of a buffer underflow, the playout freezes and resumes again upon the reception of the next frame. Not dropping late frames makes the offset between encoding and decoding times variable. In fact, in the absence of losses in the network, the offset increases with each underflow by an amount equal to the duration of the underflow.

We operate each one of these playout policies in a P2P streaming system with the following characteristics: (1) hierarchical structure, (2) threshold-based handoffs (change of upstream parent) based on partial or full information on the remaining network, (3) single-description coding. Such a setting resembles initial P2P streaming systems like the one presented in[3] and was chosen due to the popularity of such systems, their simplicity, and most importantly, in order to protect our evaluation of playout from issues that are orthogonal to it. In a sense, our chosen setting is the most fragile one as it includes a minimum amount of redundancy. Certainly one can design an over-provisioned system based on a dense overlay graph with multiple reception points and elaborate coding, but this would obscure the effects of playout policy which is what we want to isolate in this work.

We develop a simulation environment for the above policies and setting and use it to compare them across different levels of network load and heterogeneity with respect to link capacities. Our evaluation is based on "direct" metrics like *Discontinuity* and *Loss* (to be defined precisely later). To explain the observed results on these metrics we introduce a new "indirect" one – called *Availability* – which roughly amounts to the number of available upstream parents to which a node can perform a smooth handoff at a time of poor reception quality from its current parent. Based on several simulation scenarios for our control variables (load, heterogeneity, information on remote nodes, number of past frames kept) we arrive at the following observations and conclusions:

- Sync performs consistently better than Async with respect to both Discontinuity and Loss under a wide spectrum of load and heterogeneity. The improved performance can be explained by the fact that Sync maintains higher Availability and thus is able to perform smooth handoffs at times of poor reception. Under Async, the underflows contribute to the time divergence of playout points and the de-correlation of buffer contents. Thus when a node seeks a handoff it becomes difficult to find a parent with the missing frames for a smooth transition.
- Sync is effective even under limited knowledge of remote nodes (used for performing handoffs). Having the playout nodes nearly synchronized means that any one of them can offer more or less the missing frames, so we don't need to have a global view of buffer contents – tracking a small set of alternative parents suffices for handoff operations. Async on the other hand needs to know the buffer contents of remote nodes so as to identify the one (if any) whose playout point is at the right distance for a gapless handoff.
- Similarly, Sync is relatively immune to constraints on the number of downstream nodes that a parent can support. Having the playout points of different nodes near in time creates a natural load-balancing with respect to the handoffs because all nodes hold approximately the same frames thus are equally good from the standpoint of a seeking node. Contrary to this, in Async there are many cases where few nodes exist that are at the "correct" time distance from many other nodes, but cannot accommodate all of them due to these constraints and thus the seeking nodes are forced to perform handoffs that induce gaps in playout.
- Although rather counter intuitive, Async's performance is favored by randomness in parent selection (imposed by restrictions such the ones described above) since the latter eventually assists in keeping playout points "near-in-time"; peers are forced not to diverge a lot by performing handoffs that induce loss and thus restore up to a point their offset.
- Unlike Sync, Async can benefit from keeping frames in the buffer even after they have been displayed locally. This,

however, leads to several known complications (how much of it is needed to smooth out the disruption without making the offset exceedingly large) as well as some new ones (copyright restrictions permit the nodes of P2P streaming systems to buffer only a limited time window of a copyright protected material[4]).

All the above point to that Sync is a better option for the considered P2P systems. At the core of its advantage is that it is conforming to the P2P character of the application. Async on the other hand, by virtue of the divergence that it fosters, goes against the P2P paradigm by effectively reducing the number of secondary service points that are available to a node. Admittedly, an extensive field test with a modified existing system is needed to firmly verify our intuition and simulation results presented here.

Related work: Several application-layer multicast systems have been proposed for addressing the low deployment of network-layer multicast. Initial proposals mimicked the multicast and adopted a single tree topology.[5–7] The first wave of improvements to these systems aimed at addressing the unreliability of end-nodes and at decreasing the control overhead.[8,9] Additional improvements aimed at balancing the forwarding load and leveraging bandwidth heterogeneity.[10–12] Latest proposals like CoolStreaming[13] and PRIME[14] have abandoned trees in favor of BitTorrent-like[15] transmission based on swarming on top of a mesh topology. In these systems the stream is broken into different blocks and nodes obtain such blocks from multiple senders. "Buffer maps" are used for advertising the availability of blocks at each node.

All these works focus on overlay construction and coding but do not look at the details of playout scheduling. In our work we examine playout scheduling in jitter-prone environments and "fragile" tree topologies. In doing so we want to substantiate the maximum damage from using the wrong playout policy in such a setting.

We are aware of only two works directly related to ours. In[16] the authors state that for gapless playout, peer selection should not only be done based on network quality criteria, but also on the buffer status of the candidate parent peer, effectively recognizing the phenomenon of "negative correlation". However they do not associate this phenomenon with the level of synchronization between different playout schedulers, which is the main contribution of our work. In[17] different receivers achieve different synchronization levels with the source as a result of the initial prefetching mechanism. In order to improve a peer's "liveness", playout rate is slightly altered while a parent and a client peer may switch roles if the selected parent is behind in playback to facilitate catch up of the late peer. The connection between the synchronization of different receivers and their ability to cooperate by serving missing frames (what we call "availability" here) is not made.

The remainder of the article is structured as follows. In Sect. 2 we present the details of the considered playout policies and P2P setting. Several simulation scenarios on multiple metrics and control parameters are presented in Sect. 3. Finally Sect. 4 summarizes our findings and points out some interesting related observations on the operation of a popular P2P streaming system.

2. SYSTEM DESCRIPTION

In this section we present the details of the various components of our evaluation. We start with the two playout policies and move on to the details of the single tree hierarchical P2P streaming system which we consider.

Playout Policies: Let $e(k)$ denote the encoding time for the kth frame and $p_i(k)$ be its scheduled playout time at node v_i. We define the following playout schemes:

Sync(D_i): Frames that become available at peer v_i before their scheduled playout time are displayed at their exact playout time p_i. Frames that miss their playout time are skipped. This amounts to synchronous playout between the source and node v_i where by *synchronous* we indicate a *fixed offset* D_i between encoding and playout times. That is: $p_i(k) = e(k) + D_i$. Let V be the set of all peers into the system. When $D_i = D$, $\forall v_i \in V$, all nodes display the same frame at the exact same time and a global sychronization is achieved.

Async(D_i): After an initial buffering delay D_i for the first frame, subsequent frames get displayed at the earliest possible time following their previously displayed one. Assuming that no frames are lost in the network, we can define Async recursively as follows: $p_i(k) = p_i(k-1) + T + U(k-1), p_i(1) = e(1) + D_i$, where T is the nominal duration of a frame and $U(k-1)$ is the duration of a possible underflow that follows the presentation of frame $k-1$. If frame k is readily available after the presentation of frame $k-1$ then $U(k-1) = 0$.

Sync and Async stand at the two extremes of the spectrum of playout policies from delay to data preserving. Of course there exists intermediate policies in this spectrum, e.g., those that apply modified playout rates depending on the current buffer occupancy,[18] but these are rather elaborate and fall outside the scope of the current article.

Initial Tree Build-Up: We assume that nodes form a single hierarchy rooted at the video source which transmits a single-description stream. A new peer v_i selects randomly a parent peer v_j already in the system and connects to it (we discuss

reconnecting peers and handoffs later). Node v_i selects a specific frame from v_j's playout buffer and starts prefetching it and all subsequent ones for a time interval F_{ij} which leads to the desired offset D_i between the playout of this first received frame at v_i and its encoding time at the source.[19] At the end of the prefetching period, the playout process starts. It is at this point that the differences between Sync and Async start to materialize – the first one maintains this initial offset whereas the second one lets it increase by accepting and presenting late frames.

Performing Handoffs: We allow a node v_i to be in either of the following two modes:

Stable mode:A node is stably connected to its parent as long as its current buffer occupancy b_i is above a threshold value B_h and its parent hasn't left the distribution tree.

Handoff mode: A node enters a handoff mode as soon as its buffer occupancy falls beneath B_h or it is abandoned by its parent. The handoff amounts to selecting a new parent and connecting to it for a grace period T_g before returning to stable mode. The grace period allows buffer build up thus avoiding cascading handoffs.

Pre-active handoffs are employed to increase the chance for gap-free transitions when the connection to the current parent peer is not good enough, or when the latter has left the system. To perform such handoffs node v_i is supplied with a random subset $V_i \subseteq V : |V_i| = m \leq n$ (n is the number of all peers in the system) of potential parents which it keeps monitoring while in stable mode. Monitoring amounts to exchanging periodic signalling messages with each node $v \in V_i$ containing the identity of the frame currently on display at v, as well as the newest frame in v's playout buffer. To perform the handoff, v_i partitions V_i into three disjoint subsets[19] V_i^A, V_i^B, and V_i^C, and connects to a random node starting from V_i^A, continuing with V_i^B if $V_i^A = \emptyset$, and with V_i^C if both V_i^A and V_i^B are empty. V_i^A includes the known peers that have in their buffers the next missing frame for v_i, i.e., the one whose id is higher by 1 from the id of the frame that is on the top of the playout buffer of v_i (position b_i). V_i^B includes the known peers that don't have the next missing frame for v_i, but will have it in the future as their playout point hasn't exceeded it yet. V_i^C includes peers in $V_i \backslash (V_i^A \cup V_i^B)$ that hold any frames that v_i doesn't have. Peers are selected randomly within the subsets for load balancing. The handoff is gapless only when it is done towards a node of V_i^A.

3. PERFORMANCE EVALUATION

3.1 Metrics

We compare Sync(D) and Async(D) based on the following performance metrics:

Discontinuity: A discontinuity occurs when due to the unavailability of the next fresh frame(s), the last in-time rendered frame remains on display longer than its nominal time. Under Sync, the discontinuity increases by T with each frame that misses its scheduled playout time. Under Async, the discontinuity increases with each underflow, by an amount that equals the duration of the underflow. We let d denote the average discontinuity ratio expressed as the average among all peers of the total time that a peer spends viewing frozen frames to the total playback time.

Loss: Under both Sync and Async, each frame that is not displayed increases the loss by T. Under Sync, discontinuity and loss coincide. Under Async, though, the two are different because a delayed frame causes discontinuity (underflow) but not loss because it is displayed when it eventually arrives. During handoffs, the loss increases when the next missing frames do not exist on the playout buffer of the parent, in which case the scheduler starts pulling and presenting whichever frame is closer (in the future) to the next missing one. We let l denote the average loss ratio expressed as the average among all peers of the total lost playback time experienced by a peer to the total playback time of all frames that should be presented to the user.

Availability: We define the *peer availability* A to be the average (over the effective duration of the stream) ratio of compatible pairs over all possible pairs. A pair of peers (v_i, v_j) is deemed compatible when each of them may serve as a parent peer for the other without the client peer experiencing any discontinuity or loss given the following assumptions: (1) connection to a parent peer is instant, (2) the parent peer is supplied with new frames at least at the nominal rate, and (3) the available bandwidth between client and parent peers is at least equal to the nominal video rate . It is clear that due to these assumptions A is an upper bound of the real availability that can exist in practice.

3.2 Description of the Simulation Model

Initial tree formation: We assume discrete time with slot duration set equal to the frame period T. $n = 100$ peers enter the network according to a Poisson arrival process of rate 1 arrival/slot and remain in it for the entire duration of the simulation. We start collecting statistics after all peers have commenced playback.

Video source: The normal playback rate is set to 30 frames/sec, i.e., the video source at the root of the delivery tree makes available a new frame every $T = \frac{1}{30}$ seconds. Frame sizes are extracted from an educational video encoded in MPEG4

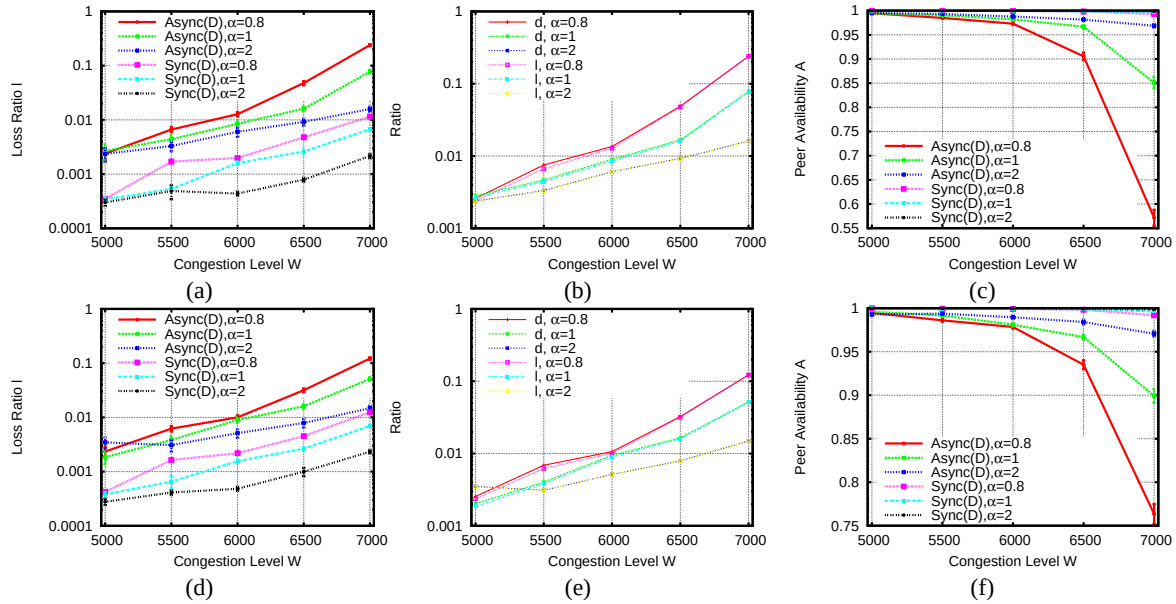


Figure 1. Results from (a)(b)(c) experiment 1,(d)(e)(f) experiment 2.

format at constant bit rate $R_N = 256$ Kbps (LectureHQ-Reisslein trace file available at [20]).

Available bandwidth of overlay links: We use a simple two-parameter model for obtaining the available bandwidth of each established overlay link: parameter W captures the *average load* (or *congestion level*) in the network, and parameter α captures the *heterogeneity* in terms of the available bandwidth of individual overlay links. We consider that at each time slot of T seconds a directed overlay link L_{ji} from a peer v_j to a peer v_i is "down" (rate 0) with a probability $P_{ji}(\alpha, W)$, which we call the "overlay link drop probability" [19] for a given heterogeneity value α and congestion level value W. At each time slot when an overlay link is "up", the value of the transmission rate is drawn uniformly at random from [R_L,R_H] Kbps. This very simple model suffices for an initial qualitative performance comparison between Sync and Async. The overall system being quite complex, it is not clear what more realistic typical workloads look like, so in the end it will take a real prototype to validate our conclusions.

3.3 Experiments

In the following we describe our simulation experiments and present our results.

The number of peers is $n = 100$, the nominal rate of the stream is $R_N = 256$ Kbps, the frame period is $T = \frac{1}{30}$ seconds, when a link is "up" its rate is drawn uniformly at random from the range $[0, 1024]$. All nodes use $D_i = D = 150 \cdot T$ seconds, i.e., they pre-buffer up to 150 frames, which is also their buffer capacity $B_c = 150$. The buffer threshold for triggering a handoff is $B_h = 10$ frames , the grace period is $T_g = 4 \cdot B_h \cdot T$ and the time between a disconnection and reconnection to a new parent peer is $T_h = 5 \cdot T$.

Our control parameters are the heterogeneity α, taking values in [0.8,2], and the average load W taking values in [5000,7000]. These ranges make a medium to high percentage of links have enough rate to support the nominal stream rate (see Fig. 3(a)). Each simulation point in our results is the average of the outcomes of 100 independent runs each of which simulated 50000 time slots of system operation. In each graph the 95^{th}-percentile confidence interval is drawn.

Experiment 1 (Unlimited upload capacity, Global information): In this experiment each peer can serve an unlimited number of downstream peers and has full knowledge of the buffer contents of all other peers i.e., $m = 100$.

In Fig. 1(a) we plot the loss ratio l against the congestion level W, for various heterogeneity values α under Sync(D) and Async(D) playout. As it may be seen, Async exhibits a much higher loss throughout the depicted α, W space. For example, for $\alpha = 0.8$ and $W = 7000$ Async has loss ratio 24% whereas Sync has only 1.2%. Increasing congestion hurts both policies as fewer links have enough effective rate (factoring in the up/down transitions) to support the stream, the damage, however, is always worse for Async. Also, for a given congestion level, the loss decreases with increasing heterogeneity, for both policies. This is expected because under high heterogeneity, very few connections have a small

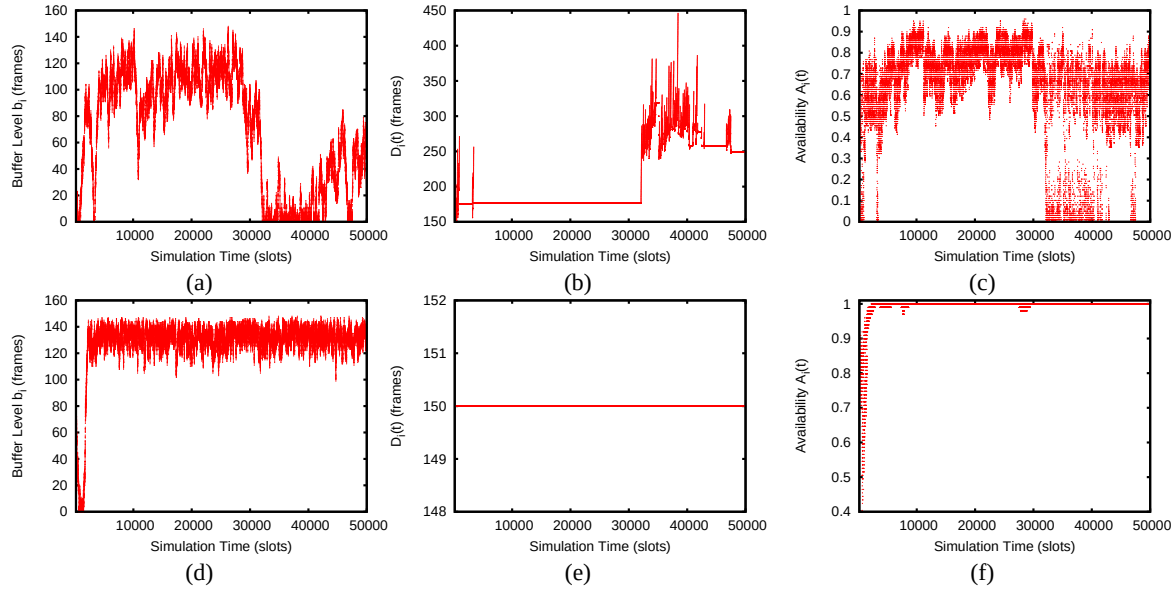


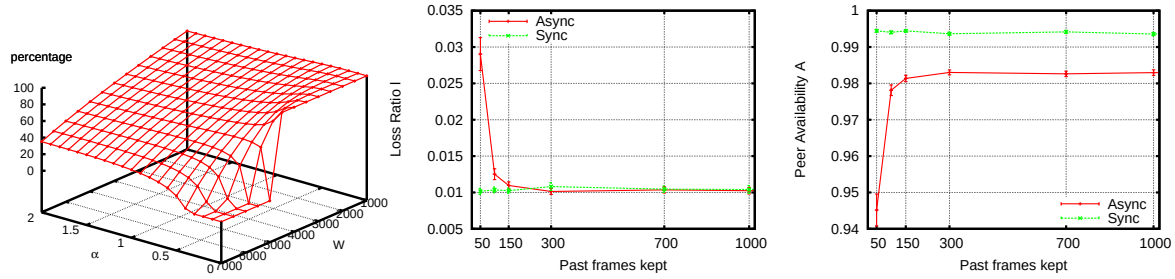
Figure 2. A single peer's buffer level, offset and availability over time during a single simulation run of experiment 1 for $a = 0.8, W = 7000$ under the (a)(b)(c) Async policy ,(d)(e)(f) Sync policy.

available rate, while the majority of them have medium to high rate[19] thus, all together, more overlay links can support the nominal rate under high heterogeneity. In Fig. 1(b) we plot the discontinuity ratio d and the loss ratio l under Async only. Both ratios seem to be pretty close (see discussion later). In Fig. 1(c) we plot the peer availability A, which we use to interpret the previous results on loss and discontinuity. Sync exhibits high availability, close to 1, which decreases slowly as the link rates fall (i.e., with higher W and lower α). Under Async, deteriorating rates de-synchronize the different playout points, as different nodes face different underflow periods. Consequently, the buffer contents become de-correlated which implies smaller availability of alternative parents which can provide for gapless handoffs. Such, gap-inducing handoffs make Async perform worse with respect to both discontinuity and loss.

To make the above more clear, we conduct a single simulation run for $\alpha = 0.8, W = 7000$ and log at every slot a single peer's v_i buffer level b_i, current offset from the source $D_i(t)$ and availability $A_i(t)$, for both policies. A peer's v_i availability $A_i(t)$ at time t is defined to be the ratio of peers which are compatible to v_i to the total number of peers. This is an upper bound to the percentage of peers that may serve as parents for v_i at time t facilitating a disruption-less handoff. In the Async case a buffer underflow (see Fig.2(a), time period 35000-40000) increases peer's offset (see Fig.2(b)) while severely decreases availability (see Fig.2(c)) since as its playout point diverges from the playout points of other peers gradually less peers are able to serve the peer with the required frames. This leads sooner or later to connection to a parent peer with loss of content, the peer has to consume later frames, which restores its offset at a lower value and availability at a higher value. This way in the Async case offset and availability fluctuate through time.
In the Sync case a buffer underflow (see Fig.2(d), time period 0-2000) does not affect the peer's offset as expected (see Fig.2(e)) while it lowers availability (see Fig.2(f)) which is kept at high levels during all times due to the fact that since playout points are the same among peers, buffer contents are highly correlated.

Experiment 2 (Limited upload capacity, Global information): In this experiment peers have full knowledge of the buffer contents of all other peers i.e., $m = 100$, but can support only a limited number of downstream peers: the source can support up to 10 peers; all other peers can support up to a number taken uniformly at random from the range [1,10] when joining the streaming hierarchy.

In Fig. 1(d) we plot the loss ratio l under Sync(D) and Async(D) playout. Sync exhibits almost the same loss ratio l as in experiment 1 (see Fig. 1(a)) and seems unaffected by the node outdegree assumed adopted in this experiment. Surprisingly Async exhibits lower loss ratio than before in experiment 1 but still remarkably higher than Sync. The same observation holds for the discontinuity ratio d exhibited by Async which takes similar values with the loss ratio l (see Fig. 1(e)). The behavior changes with α, W as described in experiment 1. In Fig. 1(f) we plot the peer availability A which is close to 1



(a) (b) (c)

Figure 3. (a) Percentage of directed overlay links that their expected rate is greater or equal to $R_N = 256$Kbps versus α and W when link rate is drawn uniformly at random from [0,1024] when a link is "up", (b) (c) The effect of the number of past frames kept at each peer observed for $\alpha = 0.8$, $W = 7000$, $B_h = 150$ in experiment 1.

for the Sync and around 0.75 for Async (i.e., Async is benefited compared to experiment 1 where it got 0.55).

The observed improvement in the performance of Async is attributed to the fact that in the current scenario, because of the limited upload capacity of peers, peers that fall behind due to underflows, do not easily find peers to which they can perform a gapless handoff. Therefore, most of the times they have to induce a gap and connect to a peer that is further ahead in time. The gap hurts momentarily, but in the long run is helpful as it leads implicitly to less divergence of playout points. On the other hand Sync exhibits almost the same performance as in experiment 1 since it manages to always retain a very high availability thus it is less likely for a peer not to find an appropriate peer to cooperate effectively.

Due to space limitation we omit experiments we conducted for unlimited upload capacity, partial information and limited upload capacity, partial information. Results from these experiments further justify those of experiment 2 and point to that *Async's performance is favored by randomness in selection since the latter assists in keeping playout points "near-in-time"*. We present these experiments along with more results for different node populations in our technical report.[19] Already presented observations and conclusions prove to be also valid when the number of peers in the system scales up.

The effect of keeping past frames: Up to now peers were assumed to be keeping only a sliding window of future frames. We now let each peer keep a number of past frames i.e., frames that have already been rendered at the local playout scheduler. We retain the terminology used till now and call "buffer" the storage space allocated for undisplayed (future) frames. We use an additional "past frames' buffer" to keep past frames. We repeat experiment 1 for $\alpha = 0.8$, $W = 7000$, $B_c = 150$ and various sizes of the past frames' buffer. In Fig.3(a) we plot loss against the number of past frames kept while in Fig.3(b) we plot the availability. One may verify that Sync is oblivious to the number of past frames kept whereas Async is favored from the existence of past frames at each peer. Loss decreases until the number of past frames kept reaches the value 300 at which point it remains flat (this amount of buffer space suffices to mask the worst kind of jitter that can appear under our on/off link model).

4. CONCLUDING REMARKS

In this work we examined playout scheduling in jitter-prone environments and "fragile" tree topologies in order to substantiate the maximum damage from using the wrong playout policy in such a setting. Below we summarize our results:

- Sync performs consistently better than Async in terms of *discontinuity* and *loss* while both policies exhibit higher performance as heterogeneity increases and congestion level decreases.
- Sync's performance is unaffected by restrictions in peer selection since it maintains high availability needing more or less only to achieve a good connection to some peer. Async's performance increases with randomness in peer selection since peers are forced not to diverge a lot by performing handoffs that induce loss.
- While Sync is oblivious to the existence of past frames at each peer, Async is highly favored. However it is not possible to know a priori how many past frames should be kept in order Async to exhibit a high performance (close to Sync's performance) since this would require knowledge of the network conditions peers will face.

Although in this work we limited to single distribution trees, we believe that our observations will also carry through to mesh-like topologies.[1] Measurement studies of popular P2P streaming systems supporting distribution over a mesh overlay topology like PPLIVE[21,22] justify our intuition. PPLIVE employs an Async alike playout policy, in which peers maintain large buffers to store recently presented chunks and chunks scheduled to be played in the future while constantly

exchanging “buffer maps”. This measurement study reports that some peers watch frames in a channel minutes behind others while at the same time large average freezing periods close to 1 min are observed. It seems that the adoption of the Sync policy in such a system would greatly improve its performance while it would eliminate the control overhead since availability would be high thus even a random selection of a list of parents would be sufficient.

ACKNOWLEDGMENTS

This work has been supported in part by the: i) IST CASCADAS program under contract FP6-027807; ii) IRAKLITOS research fellowship program, co-financed within Op. Education by the ESF (European Social Fund) and National Resources; and iii) NoE CONTENT (IST-384239).

REFERENCES

1. N. Magharei, R. Rejaie, and Y. Guo, “Mesh or multiple-tree: A comparative study of live p2p streaming approaches,” in *INFOCOM 2007*, Anchorage, Alaska,6-12 May 2007.
2. N. Laoutaris and I. Stavrakakis, “Intrastream synchronization for continuous media streams: A survey of playout schedulers,” *IEEE Network Magazine* **16(3)**, May 2002.
3. S. Rao, *Establishing the viability of end system multicast using a systems approach to protocol design*, Carnegie Mellon University, Phd Thesis, Technical Report CMU-CS-04-168, Oct. 2004.
4. D. L. Hayes, *Advanced copyright Issues on the internet*, Fenwick and West LLP, 2007.
5. H. Deshpande, M. Bawa, and H. Garcia-Molina, *Streaming live media over peer-to-peer network*, Stanford University, Technical Report, 2001.
6. Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang, “A case for end system multicast,” *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast* **20(8)**, 2002.
7. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, “Scalable application layer multicast,” in *ACM Sigcomm 2002*, Pittsburgh, Pennsylvania, August 2002.
8. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, “Bullet: high bandwidth data dissemination using an overlay mesh,” in *ACM SOSP ’03*, New York, USA, Oct. 2003.
9. D. A. Tran, K. A. Hua, and T. T. Do, “A peer-to-peer architecture for media streaming,” *IEEE Journal on Selected Areas in Communication (JSAC)* **22(1)**, January 2004.
10. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “Splitstream: High-bandwidth multicast in a cooperative environment,” in *ACM SOSP ’03*, New York, USA, Oct. 2003.
11. V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, “Distributing streaming media content using cooperative networking,” in *ACM NOSSDAV 2002*, Miami Beach, FL, USA, May 2002.
12. V. Venkataraman, K. Yoshida, and P. Francis, “Chunkyspread: Heterogeneous unstructured end system multicast,” in *ICNP 2006*, Santa Barbara, California, 12-15 November 2006.
13. X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, “Donet/coolstreaming: A data-driven overlay network for peer-to-peer live media streaming,” *Proc. INFOCOM 2005* **3**, pp. 2102–2111, Miami, FL, USA, 13-17 March 2005.
14. N. Magharei and R. Rejaie, “Prime: Peer-to-peer receiver-driven mesh-based streaming,” in *INFOCOM 2007*, Anchorage, Alaska,6-12 May 2007.
15. Bittorent, http://www.bittorrent.org.
16. C.-C. Yeh and L. S. Pui, “On the frame forwarding in peer-to-peer multimedia streaming,” in *Workshop on Advances in Peer-to-Peer Multimedia Streaming (In conjunction with ACM Multimedia 2005)*, 11 Nov. 2005, Hilton,Singapore.
17. H. Jiang and S. Jin, “Nsync: Network synchronization for peer-to-peer streaming overlay construction,” in *ACM NOSSDAV ’06*, Newport, Rhode Island, May 2006.
18. N. Laoutaris, B. V. Houdt, and I. Stavrakakis, “Optimization of a packet video receiver under different levels of delay jitter: An analytical approach,” *Performance Evaluation* **55(3-4)**, pp. 251–275, 2004.
19. C. Vassilakis, N. Laoutaris, and I. Stavrakakis, *The Impact of Playout Policy on the Performance of P2P Live Streaming*, Technical Report, Oct. 2007, http://cgi.di.uoa.gr/˜istavrak/publications/TR-2007-10-p2p-playout.pdf.
20. V. T. R. Group, http://trace.eas.asu.edu.
21. X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, “Insights into pplive: A measurement study of a large-scale p2p iptv system,” in *Workshop on Internet Protocol TV (IPTV) Services over World Wide Web (in conjunction with WWW2006)*, Edinburgh, Scotland, May 2006.
22. PPLIVE, http://www.pplive.com.