A time-shifted video streaming approach (LiveShift)

Hecht, Fabio V ; Bocek, Thomas ; Stiller, Burkhard

A Time-Shifted Video Streaming Approach (LiveShift)

Fabio V. Hecht, Thomas Bocek, Burkhard Stiller
University of Zurich, Department of Informatics (IFI), Zurich, Switzerland
Email: {hecht,bocek,stiller}@ifi.uzh.ch

I. Introduction

The growth in deployment of Fiber to the Home (FTTH) technology increases upload capacity at the edge, making the peer-to-peer (P2P) paradigm especially attractive to increase scalability and decrease cost for the publisher.

Since users of a P2P system already successfully collaborate on distributing video streams, LiveShift allows for further collaboration by having peers store received video streams in order to distribute them in the future, thus, allowing time shifting (TS) or – if the combined storage is large – even Video-on-Demand (VoD). This enables a user to watch a program from the start and jump over uninteresting parts until seamlessly catching up with the live stream without having previously prepared any local recording – all on the basis of a P2P network.

Compared to a live video streaming system, users may switch in LiveShift not only channels but also positions in a potentially large time scale. This, added to the asymmetry of interest inherent in such a scenario, demands a flexible protocol and policies that do not require peers to be simultaneously interested in data each other has.

The contribution of LiveShift is a designed, prototyped, and analyzed fully-distributed P2P streaming mesh-pull protocol and respective policies which are suitable *both* for live streaming and VoD at the same time. Many proposed P2P systems exist, which are engineered to support *either* VoD or live streaming, but LiveShift is designed for the co-existence of both. In this summary, the system is evaluated using traces from a real IPTV system to model peer behavior including channel switching. Results are measured in terms of Quality-of-Experience (QoE) metrics, such as playback lag, that are important for the end user.

II. Protocol Design

LiveShift's major protocol design objectives are the following: (1) **Free Peercasting:** Any peer is able to publish a channel, becoming a *peercaster*; (2) **Scalability:** The approach shall scale to a high number of peers; (3) **Robustness:** The system must tolerate churn; (4) **Full decentralization:** No central entities shall be present – except peercasters; and (5) **Low overhead:** Network overhead introduced must be low.

A. Segments and Blocks

Users do have the possibility of switching channels and time shifting. LiveShift adopts the mesh-pull approach [4], which adapts better to dynamic network conditions and churn when compared to tree protocols [5], by dividing the stream into chunks that are exchanged between peers with no fixed structure. Two levels of chunking are used – a *segment* is an addressing entity, which is made up of several smaller *blocks*. Each segment is uniquely identified by a *SegmentIdentifier*, which is a pair $(channelId, startTime)$ announced on the tracker by peers which offer video blocks within a segment. Blocks are small-sized, fixed-time video chunks, and are the video unit exchanged by peers.

B. Distributed Hash Table and Distributed Tracker

LiveShift uses a distributed hash table (DHT) to store the channel list and individual channel information. The DHT is responsible for storing the channel list. The distributed tracker (DT) is responsible for mapping segments to a set of providers – peers that hold at least one block in the segment. Both DHT and DT are provided by TomP2P [1].

C. Protocol Overview

The protocol is designed to allow for the implementation of different policies. A peer r, when entering the system, retrieves the channel list from the DHT. After having chosen a $channelId$ and a $startTime$ to tune into, r consults the DT to retrieve a set C_r of candidate peers that advertised blocks in the corresponding segment. Additional candidates are obtained via `PeerSuggestion` messages. Peer r then contacts a number of candidates $p \in C_r$ by sending each a `SubscribeRequest` message, containing the $SegmentIdentifier$ and a declared upload capacity.

When a peer $p \in C$ receives a `SubscribeRequest` from a peer r, it attempts to place r in its subscribers set S_p. If $|S_p| < \overline{S}_p$, the subscribers set is not full yet, and peer r is sent a `Subscribed` message, with a block map indicating which blocks in the requested segment p holds and a timeout value T_S. r will then be subscribed to receive updates to the corresponding block map via `Have` messages. If $|S_p| = \overline{S}_p$, p checks if there is another peer $q \in S_p$ that has lower priority than r (according to the policy used, *c.f.* III-A). If so, it will be preempted and removed from the set. Thus, either q or r will receive a `NotSubscribed` message. Limiting $|S_p|$ is important to control the number of `Have` messages sent.

When r receives `Subscribed`, it adds p to the neighbor set N_r and needs to verify interest periodically by computing the intersection between scheduled blocks and blocks

announced by p. If the intersection is not empty, r sends p an `Interested` message, which makes p add r in $Q_p \subset S_p$, the queue for peers waiting for an upload slot, and reply a `Queued` message, with a timeout value T_Q. On the contrary, when p has no more interesting blocks, r sends it `NotInterested` to be removed from Q_p.

Peer p has a number of upload slots $\overline{U}_p$, each of which is granted to an interested peer $r \in Q_p$. When peer r is granted an upload slot, it receives a `Granted` message. Similarly to what happens in S_p, peers with higher priority may preempt peers from upload slots.

When r is granted an upload slot from p, it is allowed to send `BlockRequest` messages to p and receive video blocks in `BlockReply` messages. This happens until either r sends a `NotInterested` message, p sends a `Queued` message (when preempted), or either sends a `Disconnect` message. Each upload slot accepts up to two `BlockRequests` at a time, to fully utilize its upload capacity, with no delays between sending a `BlockReply` and receiving the next `BlockRequest` message.

D. Peer Departure and Failure

Three mechanisms are present to react quickly to peers leaving unexpectedly or failing. When DHT routing errors exceed a threshold, the failing peer is removed from all sets, leaving space for other peers. Also, `PingRequest` messages may be used to test if peers are on-line. Peers must reply with a `PingReply` whenever they receive a `PingRequest`, otherwise they are considered failed. Finally, a peer p, when leaving, should send `PeerSuggestion` messages to all peers in S_p containing all peers in N_p as suggestions.

III. Evaluations

Evaluation results were obtained not from simulations, but by running full implementations of LiveShift. The defined LiveShift protocol is flexible, and may be used with different policies [3]. Evaluations include both channel browsing behavior and churn to produce realistic results.

A. Policies Used

The following policies have been used on evaluations and have produced good results, but further work is required to study them individually.

- Segment length: 10 minutes.
- Block length: 1 second.
- Block selection: next 15 missing blocks, at most 30 ahead of playback position.
- Block rescheduling: if `BlockRequest` takes more than twice the peer average response time.
- Candidate selection: initially 40 random peers from the DT + `PeerSuggestion` + senders of `Subscribe`.
- Neighbor selection: maximum 15, order is: (1) least amount of `Subscribe` sent, (2) highest amount of blocks received, (3) random.
- No more C_r, N_r, I_r when receiving video blocks at a rate sufficient to keep up with normal playback.

TABLE I
Evaluation Scenarios

Scenario	*Number PC*	*Number HU*	*Number LU*	*Churn*
s2	6	15	90	0
s3	6	15	120	0
s2c30	6	15	90	30%

- Members of S_p and U_p are chosen in the following order: (1) highest upload capacity, (2) highest number of blocks provided, (3) longest in queue.
- Number of upload slots: dynamically adjusted so that peer upload capacity is fully used and each slot can upload at most at the full stream bitrate.
- Timeout values: T_S is set to 5 s, and T_Q to 10 s. Inactivity timeout for peers in U is 4 s.
- Playback policy: skip n contiguous missing blocks if and only if the peer holds at least $2n$ contiguous blocks immediately afterward.
- Storage policy: storing all received blocks until the maximum capacity – 2 hours of video – is reached, then blocks with oldest download time are removed.

B. Evaluation Scenarios and Peer Behavior

Table I describes the three scenarios described in this summary. For complete evaluation results, including Scenario s1, please refer to [3]. Peers were divided in classes regarding their maximum upload capacities. While Peercasters (PC) and High Upload (HU) peers are capable of uploading at 500% the bitrate of the video stream, Low Upload (LU) peers can only upload at 50%. Peers are not limited in download capacities. All results obtained are averaged over 10 runs of 1 hour each.

The peer behavior was modeled using traces from a real IPTV system [2]. Peers are created with an inter-arrival time of 1 s and loop through the following two steps: (1) choose a channel and starting time, (2) hold to the channel, locating and downloading content from other peers. In the scenario with churn, peers, before step (1), have a certain chance of going offline for an amount of time given by the channel holding time distribution, before having again the same chance of remaining offline or going back on-line.

C. Quality-of-Experience and Scalability

The main Quality-of-Experience (QoE) metric used is the playback lag experienced by users, during holding time, from the point a tuple $(channelId, startTime)$ was selected. Figs. 1-3 show the playback lag experienced as users hold on (watch) a channel in the different proposed scenarios. The playback lag is, the difference between the time of the video block that is playing, and the time of the block that should be playing if there were no interruptions in the playback. A lower playback lag means lower start-up delay, less interruptions, and more closeness to what the user initially intended to watch, thus better user experience. It can be seen that:

a. The lag increases slowly as users continue to view a channel – this shows that users do not experience frequent stalling.

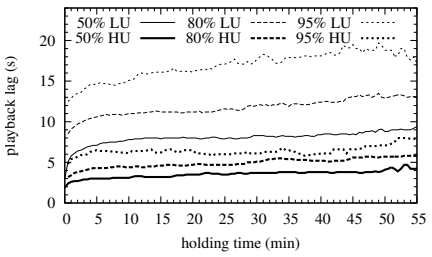

Fig. 1. Playback lag in Scenario s2

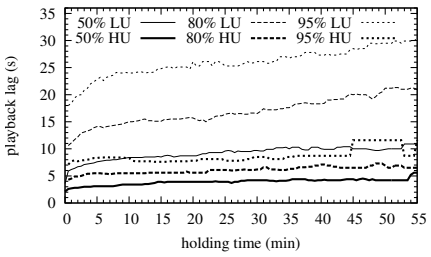

Fig. 2. Playback lag in s2c30

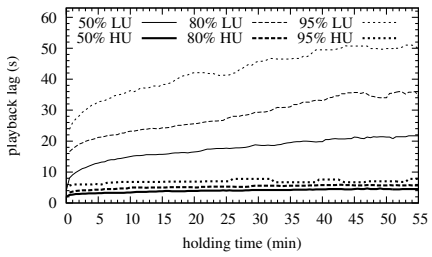

Fig. 3. Playback lag in Scenario s3

b. Even in the worst case scenarios investigated, 95% of HU peers experience lags of less than 10 s, which is an acceptable performance.
c. LU peers are more susceptible to high lag, especially in scenarios with churn or less available bandwidth. For example Fig. 2 shows 5% of LU peers with a playback lag above 25 s after watching for long periods of time, while in Fig. 3 the worst 5% of LU peers have playback lag above 50 s after watching a channel for more than 40 min.

In Scenario s3, the average distance to the peercaster increases and peers take longer on average to obtain upload slots, which are more disputed, thus, the system shows signs of being saturated, as the playback lag for several LU peers surpasses 30 s. Average playback lag is 7.70 s in s2, 14.31 s in s3, and 8.93 s in s2c30, showing that 30% churn increases the average playback lag by about 15% in s2.

D. Skipped Blocks, Failed Playback, and Overhead

According to the playback policy defined in Section III-A, some blocks may be skipped, not causing the playback lag to increase, but in fact to decrease. The share of skipped blocks is, on average, 2.41% in s2, and 1.86% in both s3 and s2c30. Interestingly, relatively less blocks are skipped in more bandwidth-constrained scenarios, due to fewer concurrent downloads happening.

The availability of content is affected by the fact that peers change their interest frequently. In the worst case, a peer may not be granted an upload slot from any of the peers which hold the blocks sought after. This may happen even when the system has spare bandwidth, due to the unbalance in content popularity – peers may have unused upload capacity due to holding only unpopular content. If the playback stalls for a long time, it is not realistic to assume that the user will wait forever. Thus, when a peer, in a sliding window of the last 30 s of playback, is able to play less than half the blocks it should, playback is considered failed, that is, the user gave up and switched to another $(channelId, startTime)$. Failed playbacks are 0.60% in s2, 1.87% in s2c30, and 4.64% in s3.

The overhead for a stream of 500 kbit/s is 3.00% on s2, 2.92% on s3, and 2.74% on s2c30, including DHT and DT traffic.

IV. Conclusions and Future Work

This summary presented a flexible and fully-decentralized mesh-pull P2P protocol for locating and distributing both live and time-shifted video streams in an integrated manner. It also sketched upon policies that can be used with the LiveShift protocol, revealing important evaluation results. These show that L supports a low playback lag for users with high bandwidth, even in the presence of churn (in form of channel switching, time shifting, and peers disconnecting). For users with more restricted bandwidth, high churn or low bandwidth scenarios negatively affect the playback, but it remains within 60 s of transmission in the investigated scenarios.

While LiveShift defines an important first step into supporting the proposed use case of supporting both live and time-shifted video streaming in a fully-decentralized environment, future work includes finding optimal policies and developing an effective incentive mechanism to verify the upload capacity of peers that may be applied in the proposed use case.

Acknowledgment

This work has been performed partially in the framework of the of the European FP7 STREP SmoothIT (FP7-2008-ICT-216259) and was backed partially by the FP7 CSA SESERV (FP7-2009-ICT-258138) coordination inputs on incentives.

References

[1] T. Bocek, "TomP2P - A Distributed Multi Map," http://tomp2p.net/, 2011.
[2] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, "Watching Television over an IP Network," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, New York, NY, USA, 2008, pp. 71–84.
[3] F. Hecht, T. Bocek, R. G. Clegg, R. Landa, D. Hausheer, and B. Stiller, "LiveShift: mesh-pull P2P live and time-shifted video streaming," University of Zurich, Department of Informatics, Tech. Rep. IFI-2010-0009, 2010.
[4] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1672–1687, December 2007.
[5] N. Magharei and R. Rejaie, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," in *Proceedings of IEEE INFOCOM 2007*, 2007, pp. 1424–1432.