Scalable Video Streaming over P2P Networks: A Matter of Harmony?

Samir Medjiah, Toufik Ahmed, Mykoniati Eleni, David Griffin

2011 IEEE 16th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)

Scalable Video Streaming over P2P Networks: A Matter of Harmony?

Samir Medjiah[1], Toufik Ahmed[1], Eleni Mykoniati[2] and David Griffin[2]

[1] *CNRS-LaBRI University of Bordeaux-1*
351, Cours de la Libération
Talence 33405,
France
{medjiah, tad}@labri.fr

[2] *Department of Electronic and Electrical Engineering University College London,*
Torrington Place, London WC1E 7JE,
United Kingdom
{e.mykoniati, dgriffin}@ee.ucl.ac.uk

***Abstract*—In this paper we address the problem of efficient layered video streaming over peer-to-peer networks and we propose a new receiver-driven streaming mechanism. The main design goal of our new layered video requesting policy is to optimize the overall distribution of video streams in terms of reliability and overhead. Since the layered peer-to-peer streaming problem is NP-Hard, we show that the classic approaches widely used in layered P2P streaming systems have some limitations and we propose an optimization technique based on harmony search which aims at increasing the rate of successful data transmissions for the most important video layers, while reducing the protocol overhead and ensuring load balancing among the participating peers. Analytical results have demonstrated that our new requesting policy enhances the streaming of layered video over mesh-based peer-to-peer networks and outperforms classic approaches.**

***Index Terms*—P2P Video Streaming; H.264/SVC; Mesh-based P2P; Pull Delivery; Optimization; Harmony Search;**

I. INTRODUCTION

Nowadays, Peer-to-Peer (P2P) video streaming has become a popular service in the Internet. P2P streaming systems such as CoolStreaming [1], PPLive [2], UUSee [3], Sop-Cast [4], and TVAnts [5] attract a lot of users. As more users get used to viewing multimedia content on-line, they will demand higher video quality than what is available with current P2P streaming systems. Providing high-quality streaming over P2P systems, however, faces multiple challenges, including: (i) the limited upload capacity of peers, (ii) high heterogeneity of receivers in terms of download bandwidth, screen resolution, and CPU capacity, and (iii) high churn rate as the peer population is constantly changing. Addressing these challenges requires not only increasing the capacity of peers and deploying additional seeding servers to make up the shortage in resources, but also employing novel methods for encoding and distributing multimedia content and developing algorithms and protocols to optimally utilize the available resources.

Most of the currently deployed P2P streaming systems, e.g., [1][2][3][4][5], do not use scalable video streams. Thus, they serve a single version of the video stream to all peers, and therefore they have limited support for heterogeneous peers. To address these issues, several studies have proposed P2P streaming systems with scalable video streams, e.g. [6][7][8][9][10]. Cui and Nahrstedt [6] presented an algorithm to decide at each receiver how to request video layers from a given set of senders. They assume that layers have equal bitrate and provide equal video quality. However, in Scalable Video Coding (H.264/SVC), layers do not have equal bitrate (spatial scalability enhancement layers may require more bandwidth than the base layer). Hefeeda and Hsu [7] studied the problem of video streaming in P2P networks for Fine-Grained Scalable (FGS) videos, taking into account the rate-distortion model of the video for maximizing the perceived quality. Rejaie and Ortega [10] presented a framework for layered P2P streaming, where a receiver coordinates the transmission of video packets from multiple senders using a TCP-friendly congestion control mechanism. The allocation of seed server resources in P2P streaming systems with scalable videos has been also considered in [9]. Lan et al. [8] proposed a scheduling algorithm for peers to request data from senders.

These papers do not consider optimal delivery of data units in terms of reliability and overhead in case of SVC content having the following characteristics: (1) Inter-layer dependency, (2) layers with different and variable bitrates, and (3) consumers may view different sub-streams of the original content. In this paper, we considered a mesh-based peer-to-peer architecture in order to distribute a layered video content (H.264/SVC). The main idea behind our contribution is the use of lightweight optimization heuristic in order to perform efficient content pulling in terms of small network overhead, reliability and load balancing. The rest of this paper is organized as follows: section II introduces harmonious requesting policy, section III presents and discusses the performance evaluation results and finally, section IV concludes the paper.

II. HARMONIOUS REQUESTING POLICY

We present in sub-section *II.A* the problem formulation of layered video delivery in a mesh-based peer-to-peer architecture. Given the information at each receiving peer (pieces to request, neighbouring peers, pieces availability …), we present a heuristic-based solution to solve the problem in sub-section *II.B*.

A. Problem Formulation

An overlay network is constructed upon an SVC stream; the overlay may be composed of different peers with different

profiles (terminal/network capabilities, user preferences …). These peers are interested in different stream scalabilities (i.e. different sub-streams). For example, a mobile device is interested in a lower spatial resolution, while a TV-set is interested in higher spatial resolution. Thus, the requested content differs from one peer to another one. *Figure 1* shows an overview of such configuration.

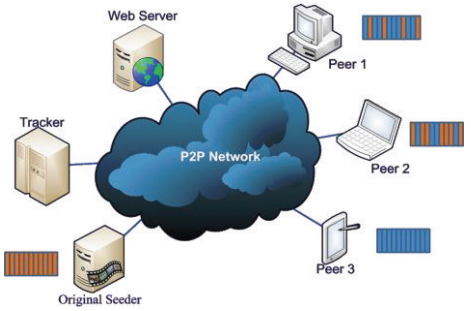

Figure 1: *Distribution of a layered video content in a BitTorrent-like peer-to-peer architecture.*

In the following sub-sections, we introduce the necessary definitions to formulate our requesting policy (i.e. how the pieces are being requested from the P2P network):

A.1. Layer-encoded Stream:

In our architecture, we have considered an SVC video to be streamed over the P2P network. The H.264/SVC format introduces three types of scalability:

- *Spatial scalability*: by layering image resolution,
- *Temporal scalability*: by layering the frame rate,
- *Quality scalability*: by layering the image compression ratio.

Thus, any combination of the three scalabilities is called a "*layer*". The first combination of the lower scalabilities is called the Base Layer (BL) while the other layers (i.e. other combinations) are called Enhancement Layers (ELs) and may depend on lower layers in order to be decoded.

Regarding the SVC bitstream, it can be seen as a collection of atomic data pieces named Network Abstraction Layer Units (NALUs).

Figure 2 shows a 3D representation of an SVC stream. In this figure, we can see that the stream contains three spatial resolutions (*QCIF*, *CIF*, *4CIF*), four temporal resolutions (*3.25*, *7.5*, *15* and *30* fps) and three quality levels (*low*, *medium*, and *high*). From this stream we can view any combination of the three types of scalability. For example, the NALUs that are shown in dark represent the necessary NALUs to extract in order to decode a CIF resolution stream with 7.5 frames/s and having a medium image quality.

A.2. Layered Video Pieces:

A stream map is a data structure that describes the entire SVC stream and it contains all the necessary information about all the NALUs that form the bitstream. Each piece π_j is described by its offset o_j in the overall stream, its size s_j, and the index of the layer l_j it belongs to (i.e. $\pi_j=(o_j,s_j,l_j)$).

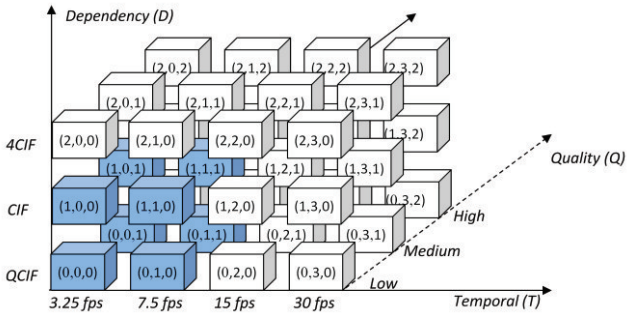

Figure 2: *3D representation of an SVC bitstream.*

In the other side, the buffer map is a data structure that describes the availability of the different pieces (as described in the stream map) at a local peer. The buffer map is more likely a set of flags that indicate whether the peer has the corresponding NALU or not.

A.3. Peers:

In our architecture, each peer maintains a set of peers $\Pi = \{p_1, p_2, p_3, \ldots, p_N\}$ serving the video stream. Each peer p_i is associated with information used to estimate its reliability r_i to deliver the video stream. This information includes: (1) the similarity of its buffer map with the local one, (2) the ratio of satisfied requests so far and (3) information about its uplink capacity.

A.4. Pieces Availability:

When a peer connects to the overlay to begin downloading the stream, it exchanges a buffer map that indicates the availability of the pieces described in the stream map. Considering a stream that contains K pieces, then the buffer map is as follows:

$$\{b_j, 1 \le j \le K\}; b_j = \begin{cases} 1 & \text{if the piece is available} \\ 0 & \text{else} \end{cases}$$

All the received buffer maps from the peers that belong to the same overlay will form a matrix $(b_{ij})_{1 \le i \le N; 1 \le j \le K}$ (where b_{ij} indicates the availability of the piece π_j at the peer p_i) that we will call further the *Piece Map*.

The Piece Map differs from the stream map. The former describes the availability of data pieces (NALUs) among the peers for the current streaming session while the latter describes the stream organization in terms of number of layers, index of each NALU, size of NALUs, etc.

The main goal of our architecture and, especially of our new requesting policy, is to achieve efficient requests in the network. By efficient requests, we mean: (1) requesting the important pieces (i.e. pieces with a lower layer index) from reliable peers, (2) reducing the overhead introduced by multiple-piece requests, and (3) load balancing the requests of different layers from different peers.

The SVC Streaming over P2P problem (SSP Problem):

Given a set of pieces to request from other peers with multi-pieces requests, the SSP problem is to *"pack"* these pieces into a minimum number of multi-piece requests while optimizing the following metrics: (1) the reliability of all the requests, (2) the overhead induced by the multi-piece requests, and (3) the load balancing of requests among the senders.

Theorem*:* The *SVC streaming over P2P problem* (*SSP Problem*) is NP-Hard.

Proof: First we show the problem is NP. The best set of requests can be found by going through all the possible requests sets, considering only the feasible ones, and assessing the "*goodness*" of each set in order to choose the best among them in a polynomial time. Next we show the problem is NP-Hard. The *Bin Packing Problem (BP)* can be reduced to the SSP problem. Given a set of items $I=\{i_1, i_2, \ldots, i_N\}$ of different packing cost c_i (volume, size, weight…) and an infinite set[1] of bins $B=\{b_1, b_2, \ldots\}$ of capacity C each, a BP formulation is to pack all the items in a way that minimizes the number of used bins. The SSP problem can be seen as a constrained variant of the classical BP problem. Indeed, the SSP formulation is to *"pack"* pieces p_i of different sizes s_i into a minimum set of requests while aiming at the same time to optimize other objectives (i.e. $R:$ *Reliability*, $LB:$ *Load Balancing* and $O:$ *Overhead*). The problem instance reduction is in fact done by considering an overloaded version of the objective function. In the classical formulation of the BP problem, the objective function used to evaluate an *"assignment"* A is the number of used bins U (i.e. $f(A)=|U|$), while in the SSP problem, the objective function considers other parameters in addition to the number of *"bins" (i.e. requests in the SSP problem)*, in other words $f(A)=f(U,R,LB,O)$. Above, we showed that BP can be reduced to our problem. Therefore, the SSP problem is also NP-Hard.

B. Harmony Search Heuristic

Given the *NP-Hardness* of the SSP problem, we propose a heuristic based on Harmony Search (HS) [11]**.** The idea behind the original HS heuristic is the process of searching a perfect state of musical harmony where each musician in a group plays a note to find the best harmony all together. The Harmony Search meta-heuristic exhibits the following properties:

- HS can consider both continuous and discontinuous objective functions.
- HS can handle discrete as well as continuous variables.
- HS is free from divergence and may escape local optima.

[1] Actually, the cardinality of the subset of B that will be used to pack the items in I, is bounded to the number of items in I, i.e., **|*B*| <= |*I*|.** In other words, for the worst case, each item is packed into a different bin

Along these properties, the harmony search heuristic is known to be a quick optimization technique in terms of necessary iterations to converge to the optimum. Figure 3 shows the harmony search heuristic applied to the SSP problem. Harmony Search algorithm is known to be free from parameter setting problem. In fact, the default values [11] are often of good choice.

Harmony_Search_for_SSP (SSP Data)

SSP inputs:
- ***PieceSet***: a set of data pieces to request (i.e. NALs)
- ***π.Peers*** / ***π*** in ***PieceSet***: a set of peers having piece ***π***
- ***Piece-Map***: a matrix for the availability of the different pieces at the different peers.

HS inputs (default values):
- ***HMS****:* Harmony Memory Size (default : **20**)
- ***hmcr****:* Harmony Memory Choosing Rate (default: **0.80**)
- ***par****:* Pitch Adjustment Rate (default : **0.20**)
- ***Termination Criterion:*** (default: **10** *iterations*)

1. Generate the *set* ***HarmonyMemory*** of *size* ***S***:
   foreach *solution* ***s*** **in** ***HarmonyMemory*** **do**
     foreach *piece* ***π*** **in** ***s.Pieces*** **do**
       Assign *piece* ***π*** to randomly chosen *peer* ***p*** in the *set* ***π.Peers***.
2. Harmony Search Algorithm:
   repeat :
     Generate a new *solution* s^*:
     foreach *piece* ***π*** **in** ***PieceSet*** **do**
       with *probability* ***hmcr*** **do**
         Assign *piece* ***π*** to the same *peer* ***p*** in a randomly chosen *solution* ***s*** from ***HarmonyMemory***.
         with *probability* ***par*** **do**
           Assign *piece* ***π*** to a *peer* p^* *"similar"* to *peer* ***p*** within the *set* ***π.Peers***
         endwith
       endwith
       with *probability* ***1–hmcr*** **do**
         Assign *piece* ***π*** to randomly chosen *peer* ***p*** in the *set* ***π.Peers***.
       endwith
     endforeach
       Compare the new *solution* s^* to the worst *solution* s^∞ in ***HarmonyMemory***:
         if (*solution* s^* *"is better than" solution* s^∞) **then**
           Replace *solution* s^* by *solution* s^∞ in the *set* ***HarmonyMemory***.
         endif
     until (*termination criterion* is satisfied).

Figure 3*: Harmony Search algorithm for the SSP problem.*

B.1. Solution Coding

A solution to the SSP problem is a set of requests. Each request is a collection of information including the peer to which it is sent, and the different pieces to request. To apply the harmony search algorithm to the SSP problem, we need a vector representation of the solution. Thus, we introduce the following definitions:

Given a set of K pieces to request, $S=[s_1, s_2, \ldots, s_K]$ is a solution, where s_i represents the peer from which the piece π_i will be requested. Since all the peers do not have the same pieces, each s_i takes a value in a different set rather than the global peers set P. We denote this set as $\pi_i.Peers$. It represents the peers that have the piece π_i.

Based on a solution vector S, the algorithm will construct the requests with respect to the size constraint C (for example, $C = 16KB$ is the optimal response size in BitTorrent Protocol as described in [12][13]).

B.2. Solution Fitness

In order to measure the fitness of a SSP solution *f(s)*, we compute the following criteria:

a) Requests Diversity:

First, the diversity of a single request is computed by eq. 1.

$$d(Req_i) = \sum_{\pi_j \in Req_i} (l_* - l_j)^2 \Big/ (\alpha L - l_j) \qquad (Eq.\ 1)$$

Where l_* is the dominant layer index in the request Req_i (for example, l_* related to the request ***Req*[1,2,3,2,2,3,2]** is $l_*=2$). L is the highest layer index of the SVC stream and α is a constant set to avoid a zero dominator (any α≠1). This indicator computes the *"distance"* between a layer index of a piece l_i and the dominant layer index l_* within a particular request, in a way that a larger difference will result in a larger "distance". Moreover, this distance is not the same for two layer indexes l_i and l_j surrounding l_* (i.e. $l_i < l_* < l_j$).

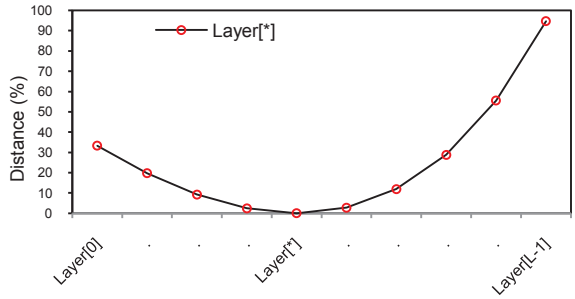

Figure 4: *The diversity "distance" for a certain layer l* and the other layers in the scalable stream.*

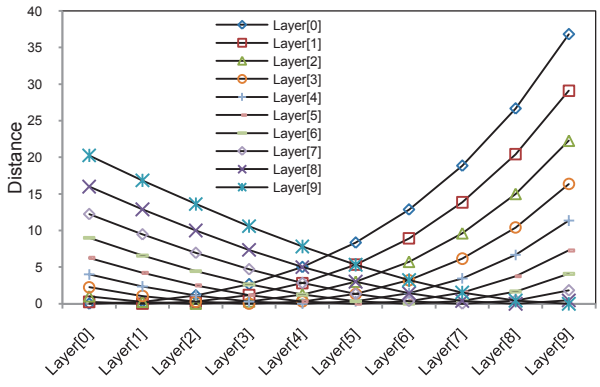

Figure 5: *The diversity "distance" for all the layers in the 10-layers stream.*

We defined this indicator in a way to tolerate diversity with lower layer indexes rather than with higher layer indexes. A plot of this indicator is given in Figure 4.

Figure 5 shows this indicator regarding a scalable stream composed of 10 layers. The diversity indicator has the following properties:

a) The diversity between l_* and itself is null.

b) The diversity is proportional to the lag between l_* and a layer index l_i.

c) The diversity is less important with lower layers than with higher layers.

While the first two properties are obvious, the third property is driven by the fact that a peer having pieces of layer l_* is more likely to have lower layers ($l < l_*$) than higher layers ($l > l_*$).

Second, the overall diversity of a SSP solution *D(s)* is defined as follows:

$$D(s) = \sum_{Req_i \in s} d(Req_i) \qquad (Eq.\ 2)$$

b) Requests Reliability

The reliability of a request depends inherently on the reliability of the peer to which it is sent and the priority of the layers being requested in this request. The reliability of a request is computed by equation 3:

$$\rho(Req_i) = \bar{l}_* \times r_i \qquad (Eq.\ 3)$$

Where, $\bar{l}_*$ is the priority of the dominant layer index in a request. In order to have a monotone increasing function, the priority of a layer l is $\bar{l} = L - l$ (i.e. lower layers are given higher priority values, and higher layer are given lower priority values).

Clearly, the reliability of a request is proportional to the priority of the dominant layer in the request and the reliability of the peer to which this request is sent. A plot of ρ is given in Figure 6 for a scalable stream composed of 15 layers and peer reliability ranging from 0% to 100%. This figure shows that in the ***p*** function, more importance is given to the layer priority than to the reliability of the peer. This is because the lower layers in a SVC stream are essential for its successful decoding.

The overall reliability of a SSP solution $R(s)$ is defined as follows:

$$R(s) = \sum_{Req_i \in s} \rho(Req_i) \qquad (Eq.\ 4)$$

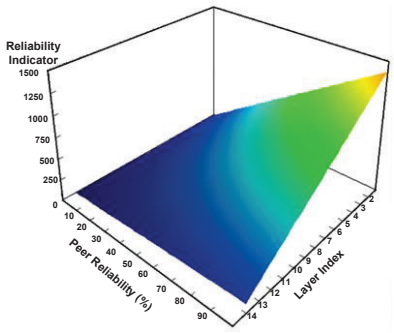

Figure 6: *The reliability indicator for a 15-layers stream*

c) Requests Overhead

In our architecture, we have defined a BitTorrent-like protocol for data exchange between the peers. Each data piece, when sent to a peer, introduces an overhead (piece index, bloc index, offset) while the overhead of a request is defined by the request index and the data size. Thus, the overhead of a request depends on the number of pieces sent in the response, since all these pieces will share the same request header.

$$\eta(s) = \frac{1}{N} \sum_{Req_i \in s} |P_i| \qquad (Eq.\ 5)$$

To model overhead, we have also considered the ratio of response filling to the optimal response size ***C***. Thus, a response to a request that minimizes the overhead will contain a great number of pieces while filling the response to its optimal size ***C***.

$$w(Req_i) = C - \sum_{\pi_j \in Req_i} s_j \qquad (Eq.\ 6)$$

So the overall overhead for a SSP solution is defined as follows:

$$W(s) = \sum_{Req_i \in s} w(Req_i) \qquad (Eq.\ 7)$$

Having the entire indicators (i.e. the criteria to optimize), we build a simple objective function that is a linear combination of all the standardized indicators: request count, request diversity, request reliability and request overhead ($\hat{N}$, $\hat{D}$, $\hat{R}$ and $\hat{W}$):

$$f(s) = \alpha_1.\hat{N} + \alpha_2.\hat{D} + \alpha_3.\hat{R} + \alpha_4.\hat{W} \qquad (Eq.\ 8)$$

A simple objective function has the coefficient equal to 1 which we have utilized in our requesting policy. These coefficients can be managed to give more importance to certain parameters.

III. Performance Evaluation

In order to evaluate the performance of our new requesting policy, we carried out some simulation experiments, using a real data set. The data set is a snapshot of a P2P network streaming a real SVC file. Table 1 summarizes the different parameters.

Data Set Summary	
Number of Layers	14 Layers
Number of NALs	7030 NALs
Reception Buffer Size	~60 KB
Total Video Length	~60sec

Table 1: *Data Set Information*

The entire piece set was organized into small NALU subsets (according to the reception buffer size). For each NALU subset we run our pull algorithm (HS), and two other existing approaches: (1) the Chunk per Peer (CPP) and the Layer per Peer (LPP)[14].

In the CPP approach, the NAL units are packed into requests with no consideration of their layer; the CPP is equivalent to the Best-Fit method in the classical Bin Packing Problem [15]. In the LPP approach on the other hand, the NALUs are packed into requests according to their layer and each request contains only NALUs of the same layer. Figure 7 shows an example of the generated requests for the three pull approaches.

Figure 7: *Generated requests for the three pull approaches.*

For each NALU subset, the three algorithms: HS, CPP, LPP output a set of requests considered as a *"solution"*. This solution is then evaluated by computing its overall fitness (i.e. score) which is based on the selected optimization indicators (Request Count, Request Diversity, Request Reliability, and the Request Overhead). The overall objective function (i.e. solution fitness) is a linear combination of the different standardized indicators ($\hat{N}$, $\hat{D}$, $\hat{R}$ and $\hat{W}$**).** These indicators were transformed to be always maximized (i.e. to minimize the number of request N, we maxmize $\hat{N}$).

Figure 8 shows a plot for the objective function for the three approaches. We can see clearly that our pull algorithm outperforms the other two algorithms since the HS approach optimizes conjointly all the criteria at the same time unlike the two other approaches.

Then, we have separately studied each criterion of the objective function; Figure 9 shows a plot of the number of generated request per NALU subsets for the three approaches. We can see in this figure that our approach generates a number of requests that is closest to the optimal number of requests achieved here by the CPP approach. Since the CPP approach is more likely a file download, the number of its requests is the smallest that we can have (see Figure 7). From this figure, we can also see that the LPP gives poor results. Indeed, when the peers are specialized to different layers, the number of requests will considerably grow since each request must have the same layer NALUs and requests for NALUs in different layers cannot be merged.

Figure 10, shows a plot of the request reliability indicator. In this figure we can see that LPP has the best performance since each request contains only NALs of a single layer. This results in a greater reliability when requests for lower layers are sent to more reliable peers (as performed in the LPP approach). However, the HS approach results are quite good since the average reliability indicator in the entire data set is 0.500 (i.e. 50%) and the standard deviation is 0.096 (i.e. 9.6%).

Figure 11 shows a plot for the request overhead indicator for the three approaches. In this figure, we can see that the overhead introduced by our approach is no significantly greater than the minimum overhead achieved here by the CPP approach. This overhead can be explained by the fact that, when trying to form requests in order to optimize all the criteria at the same time, NALUs can be assigned to a request without filling it to its optimal size (i.e. C=16KB), increasing

thus the associated overhead. However this overhead is still smaller than the overhead produced by the LPP approach

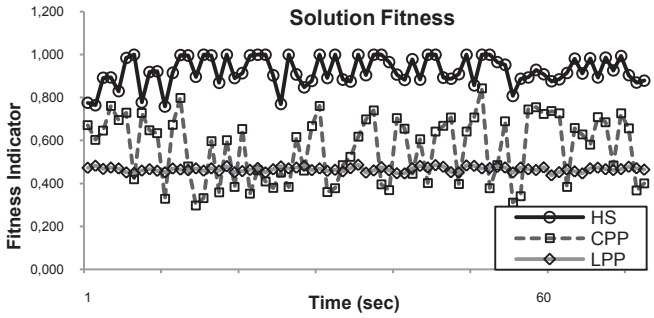

Figure 8: *Objective Function Value per NAUL subset for the three pull approaches.*

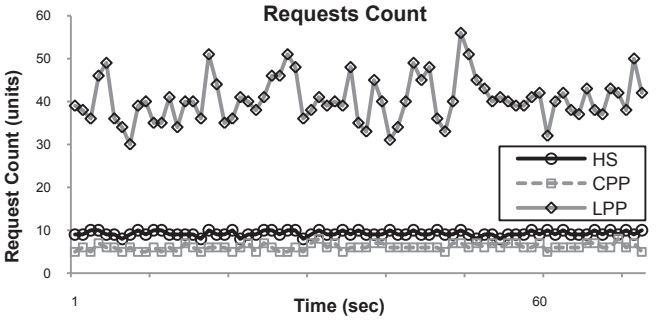

Figure 9: *Request count per NAL subset for the three approaches.*

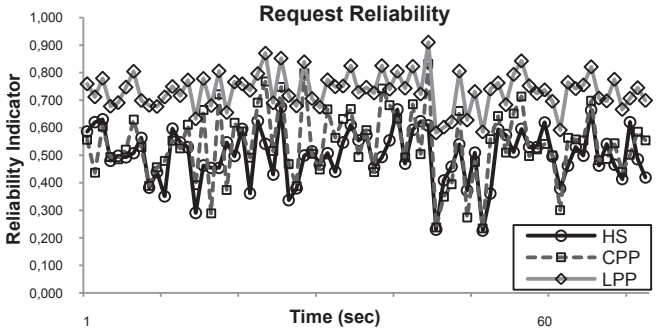

Figure 10: *Request Reliability Indicator per NAL subset for the three pull approaches.*

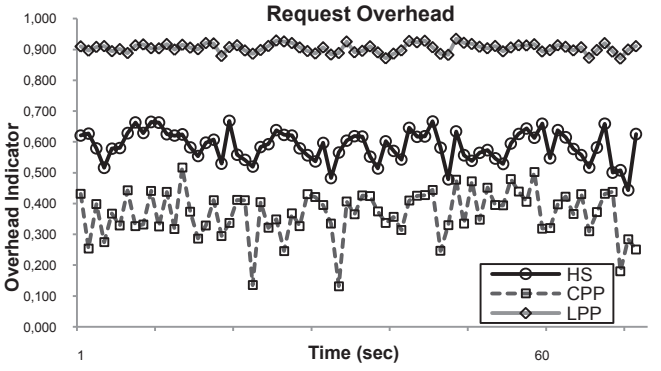

Figure 11: *Request Overhead Indicator per NAL subset for the three pull approaches.*

IV. Conclusion

In this paper, we proposed a new pull approach (HS) for efficient layered video content (e.g. SVC bitstream) distribution in a mesh-based peer-to-peer architecture. The proposed scheme optimizes different distribution criteria such as request diversity, request reliability, the number of requests and the request overhead. The performance evaluation of HS shows comparative results against two existing approaches.

Acknowledgement

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) in the ENVISION project, grant agreement 248565.

References

[1] PPLive. http://www.pplive.com/
[2] SopCast. http://www.sopcast.com/
[3] TVAnts. http://www.tvants.com/.
[4] UUSee. http://www.uusee.com/.
[5] X. Zhang, J. Liu, B. Li, and T. Yum. *DONet/CoolStreaming: a data-driven overlay network for live media streaming.* In Proc. of IEEE INFOCOM'05, pages 2102–2111, Miami, FL, March 2005.
[6] Y. Cui and K. Nahrstedt. *Layered peer-to-peer streaming*. In Proc. of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03), pages 162–171, Monterey, CA, June 2003.
[7] M. Hefeeda and C. Hsu. *Rate-distortion optimized streaming of fine-grained scalable video sequences.* ACM Transactions on Multimedia Computing, Communications and Applications, 4(1):1–28, January 2008.
[8] X. Lan, N. Zheng, J. Xue, X. Wu, and B. Gao. *A peer-to-peer architecture for efficient live scalable media streaming on Internet.* In Proc. of ACM Multimedia'07, pages 783–786, Augsburg, Germany, September 2007.
[9] K. Mokhtarian and M. Hefeeda. *Efficient allocation of seed servers in peer-to-peer streaming systems with scalable videos.* In Proc. of IEEE International Workshop on Quality of Service (IWQoS'09), pages 1–9, Charleston, SC, July 2009.
[10] R. Rejaie and A. Ortega. *PALS: peer-to-peer adaptive layered streaming*. In Proc. of ACM International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03), pages 153–161, Monterey, CA, June 2003.
[11] Z.W. Geem, J.-H. Kim and G.V. Loganathan, *A new heuristic optimization algorithm: harmony search*, Simulation **76** (2001) (2), pp. 60–68.
[12] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler, *"Small is not always Beautiful"*. In Proceedings of IPTPS, 2008
[13] *BitTorrent Protocol Specification (Version 1.0),* Link: http://wiki.theory.org/BitTorrentSpecification, 2010.
[14] Zhengye Liu; Yanming Shen; Ross, K.W.; Panwar, S.S.; Yao Wang, *"LayerP2P: Using Layered Video Chunks in P2P Live Streaming,"* Multimedia, IEEE Transactions on, vol.11, no.7, pp.1340-1352, Nov. 2009.
[15] D. S. Johnson. *Fast algorithms for bin-packing.* J. Computer. System Science, 8:272–314, 1974.