Optimized Upload Strategies for Live Scalable Video Transmission from Mobile Devices

Optimized Upload Strategies for Live Scalable Video Transmission from Mobile Devices

Matti Siekkinen, Enrico Masala, *Senior Member, IEEE,* and Jukka K. Nurminen, *Member, IEEE*

Abstract—Sharing live multimedia content is becoming increasingly popular among mobile users. In this article, we study the problem of optimizing video quality in such a scenario using scalable video coding (SVC) and chunked video content. We consider using only standard stateless HTTP servers that do not need to perform additional processing of the video content. Our key contribution is to provide close to optimal algorithms for scheduling video chunk upload for multiple clients having different viewing delays. Given such a set of clients, the problem is to decide which chunks to upload and in which order to upload them so that the quality-delay tradeoff can be optimally balanced. We show by means of simulations that the proposed algorithms can achieve notably better performance than naive solutions in practical cases. Especially the heuristic-based greedy algorithm is a good candidate for deployment on mobile devices because it is not computationally intensive but it still delivers in most cases on-par video quality compared to the more complex local optimization algorithm. We also show that using shorter video segments and being able to predict bandwidth and video chunk properties improve the delivered video quality in certain cases.

Index Terms—scalable video coding, DASH, mobile video streaming, video upload, live video, video transmission

1 Introduction

Many personal mobile devices, such as smartphones and tablets, allow capturing video content in digital format so that it can be efficiently stored, processed and transmitted. Users of such devices often desire to share the video content with friends and other people in live fashion using the wireless upload channels provided by the mobile device. Such operation is rather challenging when multimedia content is involved since the wireless channel characteristics are strongly time varying, hence it is difficult to achieve reliable low-delay communications.

Considering a generic mobile Internet access scenario, HTTP is the only communication protocol that is certainly supported in any case. Unfortunately, HTTP is notoriously ill-suited for multimedia communications. However, recently adaptive HTTP streaming techniques have been proposed to address the issue of adapting the transmission rate to the channel conditions by transferring only some segments of information at a time so that it is possible to periodically change the transmission rate if needed. The Dynamic Adaptive Streaming over HTTP (DASH) [1] standard in fact allows to describe the characteristics of the media segments and their rate, so that the best representation can be chosen at each time during the transmission.

This work focuses on an application context in which a user wants to provide the possibility to access the content, live-captured by its mobile device, in near real time using a standard stateless HTTP web server to a multitude of viewers. This is, for instance, the case of a user, present at a live event, that wants to share it with friends so that they can watch the event as it happens. Since the receivers of the communication are many, they might have different preferences. For example, a user could prefer watching with short delay while others might appreciate higher quality content despite the higher delay. In other words, each viewer has a preference for a different quality-delay tradeoff.

We base our solution on the use of HTTP over TCP to deliver video content because that allows maximal support from existing server infrastructure (e.g., CDNs). We assume that the content is captured and encoded at the mobile device. Moreover, no additional processing (e.g., transcoding or other adaptation) is performed on the server side, which enables using standard HTTP web servers which are very cost-effective. For these reasons, scalable video coding [2] is a reasonable choice, since it allows the various viewers to consume videos with different qualities without the need of re-encoding content. Scalable video coding (SVC) produces a layered video sequence where the base layer provides the most modest quality and the client may improve the quality by downloading one of more enhancement layers. Note also that scalability is compatible with the DASH standard, which is a desirable characteristic since in this way it is possible to rely on client-side adaptation without requiring any server side content processing. An additional requirement of our system is that players report their viewing delay to dedicated server(s) which in turn report them back to the mobile device that is transmitting the stream.

The challenge in the considered scenario that we tackle in this article is to optimize the video transmission process on the uploading mobile device given the variability over time of the wireless channel. The video is produced in segments and each segment has the same number of layers specified to the SVC encoder, which yields a matrix of video chunks with time and layer as the two dimensions. Hence, the problem to solve is to choose, at each time instant, which is the best video chunk among all the available ones to

- *M. Siekkinen and J. K. Nurminen are with the Department of Computer Science, School of Science, Aalto University - Finland Email: matti.siekkinen@aalto.fi, jukka.k.nurminen@aalto.fi*
- *E. Masala is with the Control and Computer Engineering Department, Politecnico di Torino, Torino, Italy - 10129, Email: masala@polito.it*

Manuscript received Sep 15, 2015.

upload to the server so that the video quality experienced by all the clients is improved over time as much as possible.

We formulate this optimization problem first with an objective to find a globally optimal solution, that is, the optimal solution given a finite length video sequence and clients with specific delays. However, this optimal solution cannot be achieved in practice, as it requires knowledge of the future bandwidth and size and quality information about video segments that have not yet been generated. Therefore, we then formulate a problem with an objective to find a sequence of locally optimal solutions, which is solvable with a real system.

We present a set of upload scheduling algorithms that solve the optimization problems and compare their performance using simulations. The results confirm that our algorithms beat naïve solutions. We also discover that the video quality delivered by a much simpler greedy algorithm comes very close the locally optimal one, which is good news as it is a much simpler algorithm to compute. However, while the locally optimal algorithm performs well, there is a clear difference to the globally optimal video quality. This result supports the hypothesis that the ability to predict bandwidth variations and video sequence information could substantially improve the video quality.

The paper is organized as follows. First, Section 2 briefly discusses the use of SVC and DASH for sharing videos captured with mobile devices. Then, an SVC-based streaming system design is presented in Section 3. Section 4 contains analytical formulation of the problem used to investigate the complexity of finding the optimal solution for the optimization problem. Practical algorithms to solve the problem are proposed in Section 5 followed by simulation setup in Section 6 and results in Section 7, where the performance is analyzed as a function of several parameters. Finally, related work in the area is presented in Section 8. Conclusions are drawn in Section 9.

2 Scalable Video Coding for DASH

Scalable video coding [2] provides mechanisms to create an embedded bitstream from which different representations can be extracted by partially decoding the compressed data. The more data is extracted, the higher the quality. In practice, the embedded bitstream features a layered structure that includes a base layer and additional enhancement layers. The base layer corresponds to the data that can be decoded independently of the other layers, providing the lowest supported quality. Additional enhancement layers improve the quality but increase the required rate and require all lower layers to be decoded.

Video scalability can be achieved by means of different techniques. The most important ones are the so-called spatial, temporal and quality (also named SNR) scalability [2]. This work focuses on the scalable extension of H.264/AVC, Scalable Video Coding (SVC) [3], which provides all these types of scalability. For simplicity, we will focus only on the SNR and spatial scalability, but the ideas underlying our approach can be extended to the other form.

The key advantage of layered coding is that the embedded bitstreams can be tailored to match the desired bandwidth by just selecting some layers and dropping others

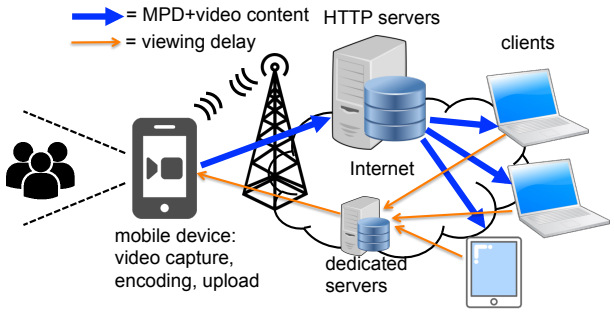


Fig. 1. System overview.

without transcoding or re-encoding the content. The SVC intrinsic support for adaptation can be exploited by standards such as DASH, which is a popular HTTP-based approach for multimedia streaming over the Internet. The Media Presentation Description (MPD) and the description of segments are covered by the DASH specification which is flexible enough to describe different resources when produced by a scalable encoder. In particular, it is possible to arrange the various layers into different dependent representations by means of a specific DASH attribute named *dependency_id*. Thus, clients can request representations as their adaptation logic requires while being sure of having all the resources (i.e., lower layers) required for decoding.

Relying on HTTP for streaming media delivery has several benefits in addition to the ubiquitous support. Above all, we mention the ability of the Internet infrastructure to efficiently support HTTP through proxies and caches [4].

3 System Design

3.1 Overview

We propose a system for live video streaming from a mobile device. Figure 1 illustrates the system and its different components considered in this work.

As soon as the user starts transmitting, a suitable Media Presentation Description (MPD) is generated and uploaded by the mobile node to a simple stateless HTTP server. The DASH MPD is parametrized (i.e., uses the SegmentTemplate tag) so that it can be kept unchanged when a new resource becomes available on the server.

Note that, to achieve low latency, the mobile device acquires and encodes video data in chunks of predetermined duration, so that they can be simply uploaded to the HTTP server and made immediately available to the viewers without any further server processing. As previously mentioned this allows to keep the system both cheap and highly scalable in terms of number of viewers.

A critical aspect of the system is how to optimally tune and configure the parameters of all subsystems so that the latency is minimized without too much performance overhead. The main focus of this work is how to deal with the typical bandwidth fluctuations of the wireless channel, which leads to the upload scheduling problem that we formulate in the next subsection. Another important parameter is the DASH segment size, whose size should be small to allow for sufficiently frequent upload scheduling decisions but large to avoid inefficiencies caused by the HTTP protocol overhead and the coding process.

3.2 Viewing Delay

Our system is designed to optimize the quality-delay tradeoff. We define the viewing delay as the time interval between the moment that the mobile device has finished encoding a particular video segment and the moment that the client player attempts to play that segment. The viewing delay can be some default delay chosen by the player software or it may be user specified, i.e. user explicitly sets the desired delay-quality tradeoff, if the player software allows it. The client player knows the start time of the stream from the MPD from which the viewing delay can be calculated.

In this work, we do not account for the variable time that is spent by the client player to download a video chunk from the HTTP server. In other words, we assume that the client can instantly fetch a chunk as it arrives to the server. In a real system, the retrieval latency is non-negligible but also unknown to the uploader. Therefore, a straightforward way to account for it would be to add a constant delay to the client reported viewing delay as a safety margin when optimizing the chunk uploading. The margin would ensure that the client manages to fetch appropriate chunks in time.

When a new video viewing client joins the system, it must transmit its viewing delay to the transmitting mobile device. The delay can be selected by the player software or it may be user specified, i.e. user explicitly sets the desired delay-quality tradeoff. In the system that we envision, the transmitting mobile device gets the client delays through dedicated servers to which the viewing clients report their delay. Note that as the amount of this information is tiny, only a small number of servers is required for a large number of clients. In other words, the video content is delivered using a common HTTP server infrastructure, whereas the delay information is obtained in a way that is specific to our system. In case a mobile client uses legacy player software that does not support reporting the viewing delay, the client can view the stream but the transmitting mobile device does not account for it when optimizing its chunk uploading strategy. Hence, such clients may experience worse video quality than those that report their viewing delays.

TABLE 1
Symbols used throughout the paper.

All times are expressed as multiple of chunks unless otherwise noted	
T_C	DASH segment duration in seconds
d	Number of DASH segments in the video
i	Segment index (starting from 0)
$i_i^{\pi_k^d}$	Segment index of the ith chunk of upload schedule π_k^d
L	Number of layers created by the video encoder
l	Layer index
$l_i^{\pi_k^d}$	Layer index of the ith chunk of upload schedule π_k^t
$\bar{l}_i$	Highest available layer for segment i
$\bar{l}_i^{\pi_k^d}$	Highest available layer for segment i after uploading chunks according to schedule π_k^d
$\theta_m^t \subset \pi_k$	Subschedule (ordered list) of generated but not uploaded chunks at t
$s_{i,l}$	Size of chunk i at layer l
$q_{i,l}$	Increment of quality of chunk i at layer l
t	Current encoding time in number of segments generated
N	Number of clients
n	Client index
δ_n	Delay of client n watching the video
Π	The set of all valid upload schedules for a video of d segments
π_k	A chunk upload schedule for a video of d segments
$S(\pi_k)$	Size of a schedule π_k, i.e., bits required to upload chunks belonging to it
$x_{i,l}^{\pi_k}(t)$	Selector variable (0 or 1) modeling the availability of chunk i at layer l
Q_n	Quality experienced by client n watching the whole video
V_k	Number of chunks to be uploaded according to schedule π_k
B_t	Number of bits that can be uploaded during segment t

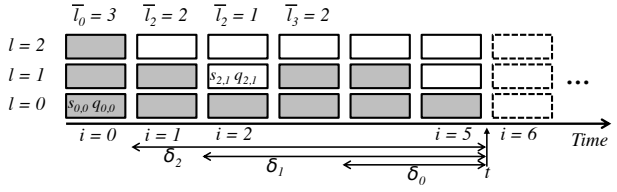


Fig. 2. Representation of chunks, split into the available layers, and the relevant time instants in the system. Chunks in gray have already been successfully uploaded to the server.

4 Upload Scheduling Problem

4.1 Problem Statement

The optimization problem can be stated as finding the upload schedule for the video chunks of the various layers already encoded by the mobile device that maximizes the expected quality for all the clients that are playing the video content. Moreover, it is of paramount importance to make sure that the base layer is always received by all clients in order to avoid annoying playback interruptions.

The schedule must take into account the dependencies between layers, i.e., it cannot upload a chunk at layer l if layer $l-1$ has not yet been uploaded. Clients are not expected to start playback at the same time. Therefore, the quality experienced by them is directly proportional to the delay, with respect to the real-time, that clients use in playing back the content. The more they lag behind real-time, the higher is the quality.

In this work we focus on maximizing the average of the quality experienced by all clients. However, more sophisticated objective functions could be used to merge the qualities of all clients in a single metric, such as maximizing the minimum of the qualities experienced by all clients, or using the minimax criterion [5].

4.2 Global Optimization Problem

In the following we analytically formulate the problem. We call it global optimization problem because we aim to find the global optimum as opposed to a local one, which we discuss in Section 4.3. Table 1 describes all the symbols used.

We refer to Fig. 2 that illustrates a generic situation in which a number of DASH segments have been encoded, each one into several layers. The current time is indicated by t and it is counted in terms of segments of video elapsed, i.e. generated. The mobile device is encoding the data for the new DASH segment which is represented by the chunks with dashed lines in the figure, and the new data will be available only after T_C seconds, i.e., the DASH segment duration. The gray chunks are the one that, given the previous upload schedule decisions and upload channel conditions, are available on the server at time t.

Let i be the index of the segment, starting from 0, produced by the encoder on the mobile device, and l be

the layer, starting from 0, corresponding to each chunk in the segment. We represent a chunk as a pair of segment index and layer (i, l). At the encoder, the size $s_{i,l}$ and the incremental contribution to quality $q_{i,l}$ of each chunk (i, l) is known, as well as how many layers, for each segment, have already been successfully uploaded to the server. This is indicated by the $\bar{l}_i$ values. As for quality, we simply measure it by computing the peak signal-to-noise ratio (PSNR) when l layers for the segment are available, as it is commonly done in the video communication research community.

Let N be the number of clients currently watching the video. Each one started playback with a certain initial delay with respect to the encoding time. Without loss of generality, we approximate such value considering the point on the segment boundary at which chunks must be available to allow playback. In other words, if the delay of the client is not a multiple of the segment duration T_C, it will be rounded toward the lowest value which is a multiple of T_C. Therefore, the situation can be summarized as in Fig. 2 where each client delay δ_n, expressed in number of segments, coincides with a segment boundary.

In this paper, we consider only the bandwidth limitations on the video uploader's side. In other words, our solution provides the optimal quality for a set of clients provided that they have enough downstream bandwidth. Mobile networks typically have asymmetric bandwidth allocations so that the upstream bandwidth is narrower than the downstream one, hence the uploader side is usually the bottleneck. This assumption also limits the computational complexity of the problem to a feasible level. The upstream bandwidth is generally not known beforehand but the uploader can measure it while uploading. B_t denotes the number of bits that can be uploaded during the time interval when the tth video segment is being generated.

We want to find the chunk upload schedule that provides the best quality for all clients among all the possible, and potentially numerous, schedules. More formally, a specific kth upload schedule is an ordered list of chunks

$$\pi_k = ((i_1^{\pi_k}, l_1^{\pi_k}), \ldots, (i_{V_k}^{\pi_k}, l_{V_k}^{\pi_k})) \tag{1}$$

drawn from the set of chunks of a video that has duration of d segments. A valid schedule cannot contain chunks of a segment for which all the lower layer chunks are not included. We denote the set of all K such valid policies for a video of d segments by

$$\Pi = (\pi_1, \ldots, \pi_K) \tag{2}$$

The size of an upload schedule π_k is

$$S(\pi_k) = \sum_{(i,l)\in\pi_k} s_{i,l} \tag{3}$$

Obviously, not all chunks belonging to an upload schedule are immediately available at the server for viewing. As the uploader progresses, more and more chunks will be available for clients. As a consequence, the larger the viewing delay of a client, the more chunks, i.e., the better quality, it will be able to play. We model the availability of a particular chunk belonging to an upload schedule π_k with a selector variable $x_{i,l}^{\pi_k}(t)$. It is one if at time instant t, which is counted in video segments from the start, the chunk (i, l) has been uploaded to the server and zero otherwise. Hence, it is a step function for each chunk belonging to the schedule and zero function for chunks that do not belong to the schedule. Using the selector variable, we can express the quality experienced by client n when watching video of d segments uploaded according to schedule π_k as

$$Q_n^{\pi_k} = \sum_{(i,l)} \left(x_{i,l}^{\pi_k}(i - 1 + \delta_n) q_{i,l} \right) \tag{4}$$

Hence, given a video of d segments, our objective is to solve the following problem

$$\begin{aligned} &\underset{\pi_k\in\Pi}{\text{maximize}}\ f(Q_0^{\pi_k}, \ldots, Q_{N-1}^{\pi_k}) \\ &\text{subject to: } S(\pi_k) \leq \sum_{i=1}^{d+\max\delta_1,\ldots,\delta_N} B_i, \end{aligned} \tag{5}$$

where $f(\cdot)$ is a function that merges together the qualities experienced by each single client into a single value. The constraint comes from the observation that it is useless to upload further chunks after all clients have played the last segment. It takes $d + \delta_n$ segments time for client n to play all the d video segments. Hence, the sum in the constraint equals the number of bits that can be uploaded until all the clients have played all the d segments. We do not set any constraints on how early a specific chunk is allowed to appear in a schedule. Consequently, if the chunk is not available at the time that it is scheduled for upload, the uploader simply waits until it has been generated.

In the rest of the paper, we set the function $f(\cdot)$ to the average of the quality experienced by all clients, i.e.,

$$f(Q_0^{\pi_k}, \ldots, Q_{N-1}^{\pi_k}) = \frac{1}{N} \sum_{n=0}^{N-1} Q_n^{\pi_k}. \tag{6}$$

The previous formulation becomes:

$$\begin{aligned} &\underset{\pi_k\in\Pi}{\text{maximize}}\ \frac{1}{N} \sum_{n=0}^{N-1} \sum_{(i,l)} \left(x_{i,l}^{\pi_k}(i - 1 + \delta_n) q_{i,l} \right) \\ &\text{subject to: } S(\pi_k) \leq \sum_{i=1}^{d+\max\delta_1,\ldots,\delta_N} B_i \end{aligned} \tag{7}$$

Intuitively, the optimal solution should maximize the video quality considering all the clients. The tricky part is to decide whether it is better to send an enhancement layer chunk of a recently encoded segment (i.e., with high i) or of an older segment (smaller i) given that clients are viewing the stream with different playback delays. A very recent chunk would usually benefit all the clients, whereas an older one will be useless to those clients that have already played the corresponding segment. As the quality increment q delivered by a chunk varies, the optimal solution needs to compare them to deem whether, for instance, an older low layer chunk with high q value is a better choice than a more recent top layer chunk with smaller q value.

Also, we assume that the upload bandwidth is enough to always accommodate the base layer (i.e., $l = 0$) of each chunk, whereas the upload of higher layers is subject to bandwidth availability. If the upstream bandwidth is less than the base layer bitrate, live video streaming is impossible and video playback at the viewer side will eventually

stall no matter which upload schedule is used. In practice, as we will see in the next section, we prioritize base layer chunks so that buffering events, which are particularly disruptive for video quality, are completely avoided given that the above bandwidth availability assumption holds.

4.3 Local Optimization Problem

Without any knowledge of the upcoming bandwidth, the global optimization problem cannot be solved. In case bandwidth prediction techniques are applicable, the global optimization problem can be solved, although in such a case the optimality depends on the prediction accuracy. Furthermore, the size and quality of chunks belonging to future video segments are generally unknown. For these reasons, it is useful to know how simpler solutions that rely on finding local optima compare to the globally optimal solution. We next formulate such a problem that does not rely on the knowledge of future bandwidth availability.

The idea is that every time a new segment has been generated, we try to find an optimal upload schedule for those chunks that have already been generated but not yet uploaded. In this way, we iteratively solve a series of local optimization problems.

Let $\theta_m^t \subset \pi_k$ denote a particular subschedule that determines a specific ordered list of already generated but not yet uploaded chunks to upload during the generation of video segment t. For example, the white blocks in Figure 2 can be members of such a subschedule. A valid subschedule is such that a combination of them forms a valid upload schedule for video of d segments: $\bigcup_{t=0}^{d} \theta_m^t \in \Pi$. Using this notion, we can formulate the problem as follows:

$$\underset{t=0,\ldots,d}{\text{for each } t\text{:}}\ \underset{\theta_m^t}{\text{maximize}}\ \frac{1}{N}\sum_{n=0}^{N-1}\sum_{(i,l)}\Big(\sum_{j=1}^{t}\big(x_{i,l}^{\theta_m^j}(i-1+\delta_n)\big)q_{i,l}\Big)$$
$$\text{subject to: } S(\theta_m^t) \leq B_t \tag{8}$$

In the above formulation, whenever a new video segment has been generated, we try to upload those chunks that maximize the quality considering all the clients. The difference to the global problem formulation (Eq (5)) is that in the above we proceed iteratively for each t, and that in each step we sum up the selector variables from all the subschedules until now, i.e., the previously chosen ones and the current tth one that we try to find, in order to determine the quality that a particular client will experience.

Note that, even in this local optimization problem, we need some estimate of B_t, i.e., the amount of bandwidth available during the time that the next video segment is being generated. We use the knowledge of the amount of available bandwidth during the previous segment as we will see in the next section.

4.4 Problem Analysis

4.4.1 Constant Size Chunks

To get some insight into the complexity of the problem, we first consider a simpler case in which all chunks have the same size, i.e., $\forall i,l: \ \ s_{i,l} = S'$. For simplicity's sake, in this analysis we assume to have a perfect knowledge of the upload channel conditions for the next segment duration (i.e., we know B_t for each value of t).

If all chunks have the same size, they can be characterized only by the quality $q_{i,l}$ they can provide to the decoded video sequence. Let us denote the number of chunks that can be uploaded by V, i.e., that is the length of the schedule that solves the problem at hand. It is obviously a function of S' and bandwidth. Let us further consider that t video segments have been encoded until now. Hence, these V chunks can be part of any of the t segments. Therefore, the number of possible combinations is the number of ways to allocate V elements in t bins. This is a well-known combinatorial problem whose solution is the so called *multiset coefficient* [6]:

$$\left(\!\!\binom{V}{t}\!\!\right) = \binom{V+t-1}{t} = \frac{(V+t-1)!}{t!(V-1)!} \tag{9}$$

Note that this is an upper bound of the number of possible combinations that need to be explored to find the optimal solution to Eq (8). In general, less than V elements can be assigned to each bin (a segment in our case) because some layers may have already been uploaded.

Considering the two flavors of the optimization problem, the number of chunks that can be uploaded is $V_{loc} = \left\lfloor \frac{B_t}{S'} \right\rfloor$ and $V_{glob} = \left\lfloor \frac{\sum_{i=1}^{d+\max \delta_1,\ldots,\delta_N} B_i}{S'} \right\rfloor$ for the local and global problems, respectively. In addition, we only need to consider the chunks of those video segments that the clients have not yet played, which reduces the solution space for the local optimization problem. In other words, t is bounded by the viewing delay. In case of the global problem, t simply equals d. In case of the local problem, it is feasible to calculate the coefficient given a reasonable number of layers. On the contrary, solving the global problem through brute force search is not computationally feasible because V and t become too large in Eq (9).

4.4.2 Variable Size Chunks - The Knapsack Problem

Video chunks are generally not of the same size. The more general problem where the chunk sizes differ can be viewed as a variation of the well-known knapsack problem. In that problem, one is given a set of items each of which has a weight and value and the task is to find the set of items whose combined value is as large as possible and which fit into a bag that can hold at most a specified amount of weight. In our case, the bag is the total amount of bits that can be transmitted, i.e., the bandwidth summed up over the time interval, and each chunk represents an item with weight being its size in bits and value its quality.

Both the local and the global optimization problems can be cast as the knapsack problem. We are specifically dealing with the 0/1 knapsack problem in which an item can be included in the solution only once or not at all. This problem can be solved efficiently using dynamic programming in $\mathcal{O}(nW)$ time where W is the bag size, i.e., the amount of bits that can be transmitted, and n is the number of chunks. Our problem presents some additional constraints: 1) a higher layer chunk cannot be included unless a lower layer chunk of the same segment is already part of the solution, and 2) the solution to our problem is an ordered set of items and the value of an item depends on its position in the

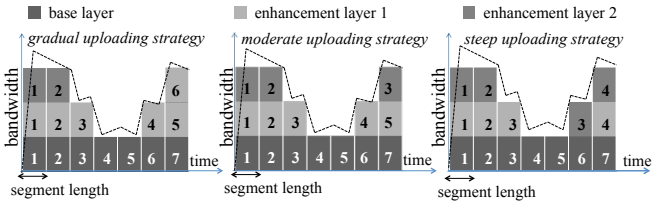


Fig. 3. Chunk uploading order for the three diagonal strategies.

bag. Neither of these constraints increases the complexity of the problem as we can handle both of them with simple comparison operations as we will show in the next section.

5 Proposed Algorithms

5.1 Naive Upload Strategies

We define three naïve strategies: *horizontal*, *vertical*, and *diagonal* strategies. These strategies are called naïve because they do not adapt to the playback schedule of clients. We define these strategies for the purposes of comparative evaluation. These strategies are summarized in the following, more details can be found in [7].

The horizontal strategy uploads all the base layer segments in order, then it continues with the first enhancement layer and so on. Therefore, it minimizes delay for all clients before enhancing quality. On the contrary, the vertical strategy uploads all layers of a video segment before moving on to the next one, hence it prioritizes quality instead of delay. It is clear that in almost all cases neither of these strategies is optimal. Instead, they can be considered as the extreme strategies in between which the optimal one lies.

The diagonal strategy lies in between the previous two. We specify three flavors of it: gradual, moderate, and steep diagonal. Common to all the three strategies is that the lower the layer, the more chunks of that layer will have been uploaded at a given point of time. In addition, the strategies adapt to bandwidth variation by uploading the base layer chunk whenever a new segment is available. Only after that, these algorithms decide on which higher layer chunks to upload before the next segment is expected to be available. The three flavors differ in the number of segments that a higher layer lags behind its immediate lower layer.

Figure 3 illustrates their upload progress. The gradual strategy spends the excess time, after uploading each of the base layer chunks, by filling up the gaps in the the first enhancement layer. The moderate strategy, instead, always uploads a chunk corresponding to a first enhancement layer if excess time remains and only after that uploads a segment corresponding to the second enhancement layer. Finally, the steep strategy ensures that the lag between the enhancement layers is at most one segment by filling up the gaps on all those layers with the same priority.

5.2 Locally Optimal Upload Strategy

We first consider algorithms for the local optimization problem. We present two variants. The first one is a greedy algorithm based on a simple heuristic. The second one is based on solving the knapsack problem using dynamic programming. Both algorithms iteratively calculate the set of chunks to upload after new segment becomes available. In this way, the algorithms adapt if the viewing delays of clients changes or new clients join and existing ones leave.

5.2.1 Simple heuristic

Our first algorithm relies on the following assumption: when uploading more layers to a segment, the quality, i.e., the PSNR, of that segment increases at a rate that is smaller than the rate at which the chunk size increases with each new layer. In other words, with each new layer, we need to transmit more bits in order to increase the quality by the same amount. Our encoding experiments suggest that this assumption usually holds. In fact, this is a consequence of the convexity of the rate-distortion curves that exhibit diminishing quality gain as the bitrate increases. Consequently, we propose a greedy algorithm that uses PSNR increase by the chunk divided by its size in bits as a heuristic to maximize upon each selection of the next chunk to upload. Because of the underlying assumption, we only need to compare the next missing chunk of each segment when making a selection. The pseudocode is shown in Algorithm 1. The algorithm is run each time a new video segment has been generated.

Algorithm 1 Greedy approach

1: input: chunk sizes and qualities $s_{i,l}$, $q_{i,l}$;
2: input: client viewing delays δ_n;
3: init: not yet uploaded avail. chunks $\mathcal{C} \leftarrow \bigcup_{l=0}^{L-1}(0,l)$;
4: init: client viewing positions $P_n \leftarrow -\delta_n$;
5: init: time till next segment is available t_{next};
6: **while** $\mathcal{C} \neq \emptyset$ **do**
7: update qualities $q_{i,l}$ weighted by P_n;
8: **repeat**
9: $W_{up} \leftarrow estimate\_upload\_budget(B_i, t_{next})$;
10: **for all** $(i, \bar{l}_i) \in \mathcal{C} : \bar{l}_i < L-1$ **do**
11: $h_i \leftarrow \frac{q_{i,\bar{l}_i+1} - q_{i,\bar{l}_i}}{s_{i,\bar{l}_i+1}}$ ▷ Compute heuristic
12: **end for**
13: $i' \leftarrow \arg\max_i h_i$
14: upload $(i', \bar{l}_{i'}+1)$ and remove it from $\mathcal{C}$;
15: $\bar{l}'_i \leftarrow \min(L-1, \bar{l}'_i + 1)$ ▷ Update uploaded layer
16: $B_i \leftarrow measure\_upload\_rate()$;
17: update t_{next};
18: **until** W_{up} too small to upload another chunk
19: wait until new segment is available;
20: add chunks of new segment into $\mathcal{C}$;
21: update P_n and t_{next};
22: **end while**

There are a few things to clarify in the code. We initialize the set of chunks available for upload but not yet uploaded to the chunks of the first segment (line 3). In other words, the algorithm starts running after the first video segment has been generated. After initialization, the algorithm enters a loop in which it stays until the video is no longer being generated and there are no more chunks left to upload that would benefit any client. In the beginning of the loop, the chunk qualities are recomputed so that the quality increase is weighted by the number of clients that will benefit from its upload, i.e., those that are currently viewing a more ancient segment. There is a second inner loop (lines 8-18) within which the algorithm selects and uploads chunks one by one based on the heuristic value until the latest upload rate measurement suggests that the remaining upload budget would no longer enable uploading another chunk before

the next video segment becomes available. After that, the algorithm waits until this happens, adds the new chunks to the set of chunks available for upload, and proceeds again to select and upload them.

Note that the amount of time spent idle waiting for chunks of new segment to become available depends, on one hand, the accuracy of the upload rate estimation and, on the other hand, the chunk sizes. If chunk sizes are large, the expected amount of accumulated idle waiting time is also larger than in the case of small chunks because the duration of the idle time is directly related to chunk sizes. We investigate the effect of this phenomenon on the performance of the algorithms in Section 7.

The complexity of Algorithm 1 is $\mathcal{O}(Vt)$ which can be slightly improved if the h_i data is not searched for maximum every time but it is kept in a structure such as a sorted heap that allows building in $\mathcal{O}(t \log t)$ time but inserting in $\mathcal{O}(\log t)$ time, operation that needs to be performed $V - 1$ times. Therefore, the final complexity is $\mathcal{O}(t \log t + V \log t) = \mathcal{O}((t + V) \log t)$.

5.2.2 Using Dynamic Programming

Our second algorithm for the local optimization problem is based on solving the 0/1 knapsack problem using dynamic programming. The pseudocode of the algorithm is presented in Algorithm 2 and of the knapsack solver in Algorithm 3.

Algorithm 2 Local optimization using 0/1 knapsack solver

```
1: input: chunk sizes and qualities s_{i,l}, q_{i,l};
2: input: client viewing delays δ_n;
3: init: not yet uploaded avail. chunks C ← ⋃_{l=0}^{L-1} (0, l);
4: init: client viewing positions P_n ← −δ_n;
5: init: time till next segment is available t_next;
6: while C ≠ ∅ do
7:     update qualities q_{i,l} weighted by P_n;
8:     repeat
9:         W_up ← estimate_upload_budget(B_i, t_next);
10:        Π ← solve_knsack(C, q_{i,l}, P_n, W_up);        ▷ Algorithm 3
11:        upload Π;
12:        remove Π from C;
13:        B_i ← measure_upload_rate();
14:        update t_next;
15:    until W_up too small to upload another chunk
16:    wait until new segment is available;
17:    add chunks of new segment into C;
18:    update P_n and t_next;
19: end while
```

Algorithm 2 only differs from Algorithm 1 in the contents of the inner loop (lines 8-15). Upload budget is estimated in the same way but instead of computing the heuristic, the algorithm calls the knapsack solver algorithm. The reason why this call is enclosed in the inner loop is that the sack size, i.e., the amount of bits that can be uploaded until the next segment is available, is an estimate. In case it gets underestimated, the loop ensures that a new solution is calculated until there is no more slack time.

Algorithm 3 is in most parts a standard 0/1 knapsack solver that uses dynamic programming. The only specificities are 1) lines 6-8 that set the quality of the base layer chunks with imminent deadline to infinity in order to surely include them into the schedule; 2) the second condition on line 17, which ensures that no upper layer chunks will be added into the schedule before all the lower layer chunks of the considered segment are included; and 3) the penultimate line, which sorts the chunks in the resulting schedule by quality increase per bit. The sorting is done because the given bit budget, i.e., sack size, may be an overestimate of the true one and sorting ensures that the most valuable chunks are uploaded first. Finally, we increment the sack size by w_{inc} at each iteration (line 16) instead of one. The reason is that the sack size is relatively large in terms of bits and bit by bit incrementing would result in too much computation. We set the value of w_{inc} empirically by avoiding too large increment that would degrade the output of the algorithm. In the simulations (Section 7), we used 50Kbit and 100Kbit as w_{inc} depending on the input video sequence.

Algorithm 3 0/1 knapsack solver for local optimization

```
1: input: set of not yet uploaded available chunks C;
2: input: chunk qualities q_{i,l};
3: input: viewing positions of clients P_n in segments from start;
4: input: number of bits W_up to schedule for upload;
5: init: Π ← ∅
6: for all (i, 0) : i − P_n < 2 do
7:     q_{i,0} ← ∞                    ▷ Ensure that base layer is never late
8: end for
9: for j = 0 to W_up step w_inc do
10:    m[0, j] ← 0;
11:    Π[0, j] ← ∅;
12: end for
13: c ← 0;
14: for all chunks (i, l) ∈ C and i ≥ min(P_n) do
15:    c ← c + 1;
16:    for j = 0 to W_up step w_inc do
17:        if s_{i,l} ≤ j and (i, l − 1) ∈ Π[c − 1, j − s_{i,l}] then
18:            if m[c − 1, j − s_{i,l}] + q_{i,l} > m[c − 1, j] then
19:                m[c, j] ← m[c − 1, j − s_{i,l}] + q_{i,l}
20:                Π[c, j] ← Π[c − 1, j − s_{i,l}] ∪ (i, l)
21:            else
22:                m[c, j] ← m[c − 1, j]
23:                Π[c, j] ← Π[c − 1, j]
24:            end if
25:        else
26:            m[c, j] ← m[c − 1, j]
27:            Π[c, j] ← Π[c − 1, j]
28:        end if
29:    end for
30: end for
31: sort Π[c, W_up] in descending order by q_{i,l}/s_{i,l}
32: return Π[c, W_up];
```

5.3 Globally Optimal Upload Strategy

We also designed an algorithm to compute the globally optimal upload schedule. Obviously, such an algorithm requires some way to predict the future bandwidth variation. In addition, the future chunk sizes and qualities need to be somehow known or estimated. We use this algorithm with an oracle that has perfect knowledge of the future bandwidth and chunk information in the evaluation section in order to understand how far the schedule given by the locally optimal algorithm is from the best possible one.

The algorithm consists almost solely of the 0/1 knapsack solver which is executed only once. We show the pseudocode of the solver in Algorithm 4. It follows the exact same logic as Algorithm 3 but there are a few important details that differ.

The algorithm needs to keep track of how much time uploading each sack would take because the chunk qualities

Algorithm 4 0/1 knapsack solver for global optimization

```
1: input: set of all chunks C;
2: input: chunk sizes and qualities s_{i,l}, q_{i,l};
3: input: viewing delays of clients δ;
4: input: bandwidth information B_i;
5: init: W_up ← Σ_{i=1}^{d+max δ_1,...,δ_N} B_i;
6: init: client viewing positions P_n ← −δ_n;
7: init: Π ← ∅
8: for j = 0 to W_up step w_inc do
9:     m[0, j] ← 0;
10:    Π[0, j] ← ∅;
11: end for
12: c ← 0;
13: for all chunks (i, l) ∈ C and i ≥ min(P_n) do
14:    c ← c + 1;
15:    for j = 0 to W_up step w_inc do
16:        update P_n and q_{i,l} given Π[c − 1, j − s_{i,l}];
17:        if s_{i,l} ≤ j and (i, l − 1) ∈ Π[c − 1, j − s_{i,l}] then
18:            if m[c − 1, j − s_{i,l}] + q_{i,l} > m[c − 1, j] and m[c − 1, j −
       s_{i,l}] + q_{i,l} > m[c, j − 1] then
19:                m[c, j] ← m[c − 1, j − s_{i,l}] + q_{i,l}
20:                Π[c, j] ← Π[c − 1, j − s_{i,l}] ∪ (i, l)
21:            else
22:                m[c, j] ← m[c − 1, j]
23:                Π[c, j] ← Π[c − 1, j]
24:            end if
25:        else
26:            m[c, j] ← m[c − 1, j]
27:            Π[c, j] ← Π[c − 1, j]
28:        end if
29:    end for
30: end for
31: return Π[c, W_up];
```

depend on time. Hence, the chunk qualities need to be updated prior to deciding whether to include a chunk into a sack (line 16). Note that the locally optimal algorithm does not need to account for this issue because the viewing positions of clients do not change during the generation (and playback) of one segment.

Because of the time dependency of chunk qualities, a chunk may deliver a worse quality when put into a larger sack than when put into a smaller size sack. In other words, the more time passes, the worse the quality of a chunk gets because there are fewer and fewer clients that would benefit from its upload. This is not taken into account by a standard 0/1 knapsack solver. We handle this issue by adding the second condition in the if statement on line 18.

Finally, the algorithm does not sort the resulting schedule as the locally optimal one does because the chunks must be uploaded in the order specified by the algorithms solution. Otherwise, the optimality cannot be guaranteed since the viewing progress by clients have been accounted for in the solution, unlike in the local algorithm.

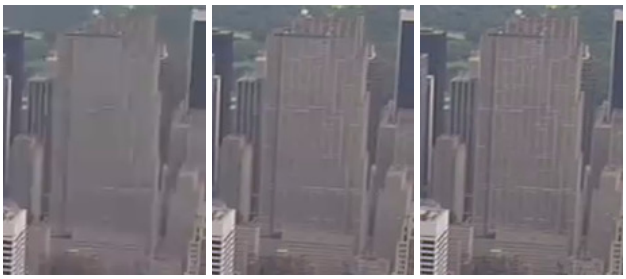
6 Simulation Setup

In order to quantify the performance of the different algorithms, we simulated the behavior of the different uploading algorithms in a number of different situations using Matlab. We used real SVC-encoded video sequence information as input to the simulations. The simulation parameters and their value ranges are summarized in Table 2.

In particular, we encoded the standard video sequences known as *city*, *crew*, *harbour*, *soccer* at 30 frames per second (fps) and 4CIF (704×576) resolution. We believe these sequences may well represent content that is live captured and

TABLE 2
Simulation parameters.

Parameter	Value range (default value, if applicable)
simulation repetitions	500
number of clients	1-100 (5)
client viewing delay	0-30 s
average uplink rate R	1-6 Mbps (2Mbps)
rate variation range	$(0.5 - 1.5) \times R$
rate change probability p_r	$0.1 - 0.9$ (0.5)
video duration	80 s
video frame rate	30 fps
video segment length	2 s
video resolutions	QCIF (176x144), CIF (352x288), 4CIF (704x576)
SVC layers	3
layer bitrates (SNR)	0.5, 1, 1.5 Mbps
layer bitrates (spatial)	0.1, 0.4, 2.3 Mbps
video sequences	city, crew, harbour, soccer
type of scalability	SNR/spatial scalability
video segment fragments	1-4 (1)



(a) Base layer (L0) (b) Enh. layer #1 (c) Enh. layer #2

Fig. 4. Visual comparison of a detail of the reconstructed *city* sequence when more layers of SNR scalability are considered. Frame #1.

transmitted to make it immediately available to viewers. We employed SNR scalability with three layers (base and two enhancements). The sequences are balanced in terms of spatial details, since *soccer* and *crew* present movements but not so many tiny details, while the opposite holds for the other two sequences. To perform encoding we resorted to the JSVM encoding software v. 9.19.15. The encoding has been configured for the DASH environment by creating 60-frame segments that can be decoded independently from the previous and subsequent segment. Each segment employs a single I frame at the beginning, followed by P frames every 4 frames. Intermediate frames are hierarchically coded B frames. The sequence is terminated by a P frame not to introduce dependencies on the next segment. The same structure is replicated for the two SNR enhancement layers. The layers have been encoded with a fixed quantization parameter in order to achieve approximately 500 kbit/s, 1000 kbit/s and 1500 kbit/s for each of the three layers, respectively. For the spatial scalability case the same coding structure has been used but three different resolutions are employed: QCIF (176×144), CIF (352×288) and 4CIF (704×576). In this case the average bitrate is about 100 kbit/s for the base layer, 400 kbit/s for the intermediate one and 2300 kbit/s for the higher one. These values increase more rapidly than with SNR scalability since each layer is expected to both handle a higher number of pixels and provide an increased visual quality.

The standard video sequences have 300 frames, therefore to generate a sufficiently long video sequence we concate-

TABLE 3
Encoding PSNR (dB) when an increasing number of SNR/spatial enhancement layers is available.

Sequence	SNR scalability			spatial scalability		
	base layer	L1	L2	base layer	L1	L2
city	31.82	34.99	36.42	34.06	34.14	36.28
crew	31.41	35.25	37.18	32.99	35.37	38.19
harbour	27.72	31.14	33.05	31.91	33.11	33.99
soccer	30.41	34.39	36.98	30.06	35.49	38.22

nated the four sequences twice (in the order *city*, *crew*, *harbour*, *soccer*), which yields a 80 s video. Table 3 reports the encoding PSNR for the various layers of each sequence for the case of SNR and spatial scalability. A visual comparison of the quality provided by the three layers is shown in Fig. 4.

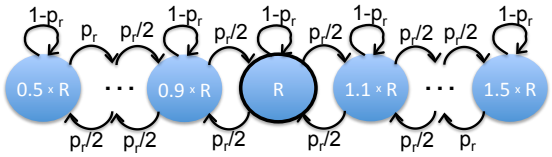


Fig. 5. Markov chain used to model upload rate variation. R is the mean upload rate.

We used a Markov chain based model of the wireless channel. The model comprises 11 states corresponding to available bandwidth of 0.5 to 1.5 times the average as illustrated in Figure 5. Such models allow capturing the graceful degradation and improvement of the channel capacity while the user is mobile and of the available bandwidth when more clients join to share the resources of the wireless network that the user is connected to. The model takes two parameters: R is the mean upload rate and p_r is the probability of transitioning from the current state to next or previous state, both transitions being equally likely (i.e., $p_r/2$). As each simulated scenario was repeated 500 times, we used at each iteration a different seed (the iteration number) for the random number generator associated with the Markov model. This manoeuvre ensures that the average upload rate measured over all the iterations does not change significantly between different simulated scenarios having different values of p_r but the same R.

The video sequences are encoded in such a way that the average bitrate of all the layers combined together is slightly below the highest average bandwidth level in our simulations, while the average bitrate of the base layer is slightly below the lowest average bandwidth level. In other words, all test cases have enough bandwidth to deliver at least the base layer to all clients regardless of their viewing delays. By selecting the test cases in this way, we make an implicit assumption that the client has some knowledge of how much the bandwidth typically varies when choosing the encoding parameters, otherwise the video playback at the viewer side will eventually stall.

Each of the simulated scenarios corresponding to a specific set of parameter values (number of clients, average available bandwidth, and bandwidth variation) was repeated 500 times and results were averaged. For each of these 500 simulations we drew uniformly random viewing delay between zero and 30 seconds for each client. These viewing delays were given to the algorithms as input parameters except for the naïve algorithms which are oblivious to viewers progress. Note that the exact same 500 sets of viewing delays were used in the different scenarios ensuring that the results are comparable.

We measure different video quality related metrics in order to compare the performance of the algorithms. The average PSNR experienced by the viewers is the main metric. The average layer played is a metric that in most cases heavily correlates with the PSNR. We also keep track of the buffering time which is the amount of time that the client's playback buffer is empty and the player must pause to wait for missing content due to unlucky scheduling decisions.

7 Performance Evaluation

7.1 Video Quality

Figure 6 plots the results in terms of the resulting PSNR for the median client with different amounts of average available bandwidth. Median client here means that we compute the average PSNR for each client of a simulated session and then take the median of those results. The whiskers in the figure correspond to the median of worst and best quality of all the simulation rounds. The state transitioning probability p_r of the Markov model was set to 0.5 and the video quality metrics are averaged over the five clients simulated. In addition to PSNR, we also analyzed the average layer played by clients. However, since they both behave similarly, we only focus on the average PSNR because it is the metric that our algorithms optimize for. As one might expect, the more bandwidth is available, the better is the video quality regardless of the strategy.

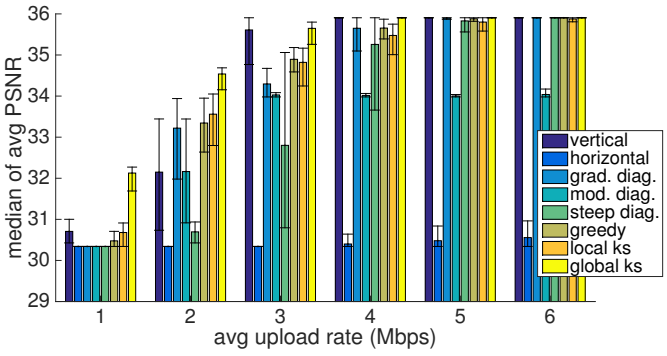


Fig. 6. Average video quality as a function of the average amount of available upstream bandwidth. Whiskers mark the median of each simulation round's best and worst qualities.

The first curious observation is that the vertical strategy delivers surprisingly high video quality. In this case, these results are a bit misleading because the vertical upload strategy is the only one that causes any noticeable amount of buffering time. Specifically, it yields a buffering ratio (total buffering time divided by video duration) of 1.7 and 0.3 for 1 Mbps and 2 Mbps average upload rates, respectively (3 Mbps is just enough to deliver all the layers). All the other strategies yield practically zero buffering ratio for all evaluated cases. Hence, the good playback quality of

the vertical strategy is undermined by high buffering ratio, which we deemed unacceptable when designing our algorithms. Among the other naïve strategies, it seems that the gradual diagonal is the most suitable for the simulated scenarios. However, the issue with any diagonal strategy is that they do not adapt to the network conditions. As explained in [8] for an equivalent diagonal download strategy, the "slope" of the best possible diagonal strategy depends on the available rate and its variation. Hence, a diagonal strategy with fixed parameters only works optimally with specific network conditions. Adjusting a diagonal strategy to the network conditions is essentially what our proposed algorithms do. In our simulations, the gradual diagonal strategy just happened to fit the simulation setup well and the moderate and steep not so well.

Obviously, the strategy that outperforms all the others in every scenario is the one based on the global optimization algorithm (Alg 4) that requires oracle knowledge about future bandwidth and chunk sizes and qualities. The local optimization with knapsack solver (Alg 2), which we call *local knapsack* (ks in the figure), delivers the second best video quality. However, the greedy algorithm performs equally well. It is even marginally better in some cases, which is probably caused by the fact that the knapsack solution is constructed using imperfect knowledge of the amount of bandwidth available to upload chunks. The results thus suggest that using the greedy algorithm, which is much simpler to compute, is a good choice in most cases. However, it should be remembered that the greedy algorithm relies on the assumption that the quality per bit decreases when adding layers. If this assumption cannot be guaranteed, it is better to use the local knapsack algorithm.

We also see that the range between the worst and best case qualities delivered by our algorithms is reasonably small, which is an indication that they deliver a relatively fair level of video quality to different clients. Note that we expect and want to see some differences between clients with different delay, because the algorithms are designed to take advantage of the the delay vs. quality tradeoff. To further illustrate this point, we plot the tradeoff for the different strategies in Figure 7 in a chosen scenario. The clients who benefit the most from the optimized upload strategies compared to the naïve strategies are those having having relatively short delay. In other words, while enabling the delay-quality tradeoff, our algorithms also deliver a more balanced video quality to the clients with different delays compared to the naïve upload strategies. If a more even quality is desired, one could design a different objective function and adjust the algorithms accordingly.

There is a clear difference between the globally optimal video quality and the quality delivered by the two strategies based on local optimization, which suggests that a capability to predict bandwidth variation, such as the solution presented in [9], and upcoming video chunk information would notably improve the video quality. We also evaluated the video quality with a version of the local knapsack that has oracle knowledge about the amount of bandwidth available during the current segment's time interval. In other words, instead of estimating the bandwidth (line 9 of Alg 2) the uploader knows exactly the upload budget. We learned that this knowledge does not substantially improve the results.

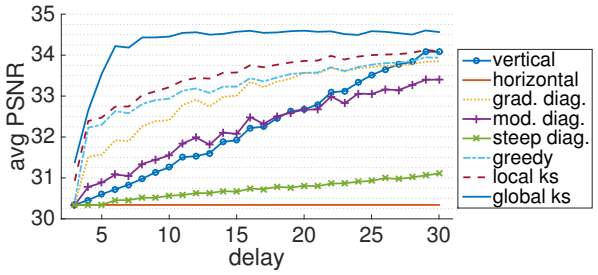


Fig. 7. Tradeoff between video quality and delay with 2 Mbps average upload rate.

Hence, so far it appears that the uploader should be able to see further in the future in order to make a difference.

We wanted to make sure that the video sequence specifics do not play a major role in the performance of the different uploading strategies. Hence, we ran the same set of simulations separately with the different video sequences. We repeated each 10s short sequence 8 times in order to obtain 80s long sequences. The simulation results are plotted in Figure 8. The results are qualitatively the same across the different video sequences. Quantitatively the results differ just because the different sequences yield different PSNR values as we saw earlier in Table 3. Hence, we conclude that the contents of the video sequence does not have a major effect on the performance of the algorithms.

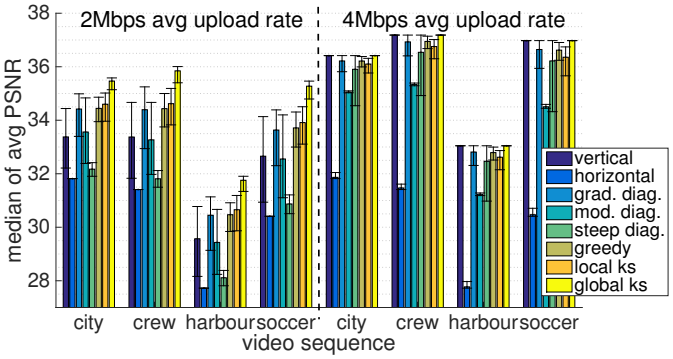
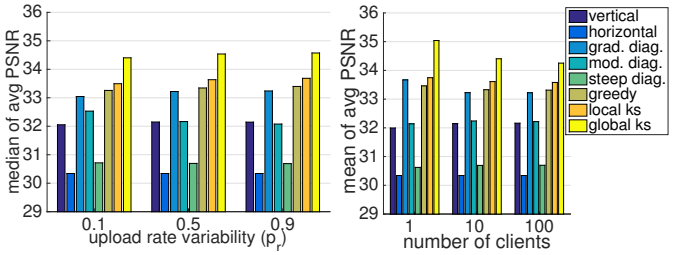


Fig. 8. Average video quality with different video sequences and 2Mbps and 4Mbps upload rates.

We next examined the impact of the rate of variation of the available upstream bandwidth and the number of clients. We changed the state transition probability p_r of the Markov model (Figure 5) and simulated the resulting video quality. The results are plotted in Figure 9. The average video quality becomes slightly worse when there are more clients in the system, which is logical as it is more difficult to optimize the quality experienced by many clients with different viewing delays than to optimize it, for instance, for just one client. But the fact that the rate of variation of the upstream bandwidth seems to have visually no effect on the results is a surprise. In fact, it undermines our hypothesis that the global knapsack performs better than the local knapsack because of its capability to "see" into the future. We uncover the reason in the next step of evaluation where we look at the role of the chunk size.



(a) bandwidth variation (5 clients) (b) number of clients (bwvar= 0.5)

Fig. 9. Impact of upload rate variation and number of clients on the video quality. The average upload rate was 2 Mbps.

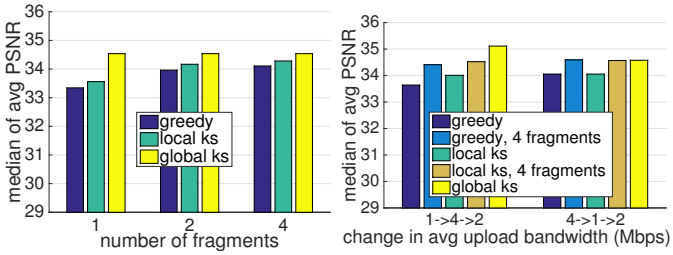


Fig. 10. Impact of fragmentation on video quality.

Fig. 11. Impact of change in average available upload bandwidth.

Before continuing with the evaluation, we wish to inform the reader about the role of the type of scalability. The results presented in the previous section were generated using a video sequence that was encoded using SNR scalability. We also investigated how the type of scalability used in the encoding influences the video quality by reproducing the results with a video sequence encoded using spatial scalability. The results were similar to the case of SNR scalability suggesting that the type of scalability plays a minor role in this work.

7.2 Fragmenting the Video Chunks

If the video is divided into large chunks, each chunk takes potentially a long time to deliver and the probability of the available bandwidth to change during the upload is greater than in the case of video divided into small chunks. Furthermore, larger chunks tend to cause larger amount of idle time because the uploader determines that it cannot upload the whole enhanced layer chunk before a new segment becomes available. Only the locally optimal algorithms are concerned with this effect since the globally optimal algorithm does not pause to wait for the next segment to be available.

Even if the video content is chunked into several seconds long chunks, each chunk can be delivered in smaller fragments. Considering HTTP-over-TCP-based video delivery, such fragmentation can be easily achieved on the HTTP layer by the uploading client. Fragmenting video chunks may increase header overhead. HTTP level header overhead can be almost completely mitigated by using, for instance, a Websocket-based uploader client. On the other hand, individual IP packets are transported by TCP. It typically uses a maximum segment size of around 1.5 kB, which imposes a limit to a fragment size below which the TCP/IP header overhead increases. However, considering that in the video sequence we use in the evaluation, the size of the base layer chunk, i.e., the layer with the lowest bitrate, is a bit more than 60 kB on the average, this overhead is insignificant unless we split chunks into more than 40 fragments.

To quantify the effect of fragmentation, we simulated scenarios where the 2 s video chunk is delivered in 1-4 fragments. Each video chunk can be played only after all the fragments have arrived, as would be the case with fragmentation taking place in the HTTP layer. In other words, individual fragments cannot be played. Similarly, the fragments of a given video chunk are available for uploading only after the entire segment has been generated.

Figure 10 shows how the video quality changes when increasing the number of fragments. Note that the global knapsack performs the same with or without fragmentation. The reason is that fragmentation presents an advantage only with local strategies because they can use the remaining idle time before next segment arrival to transmit fragments. In other words, fragmentation reduces the idle wait time but that time is already zero when using the global knapsack algorithm because it computes the entire schedule at once. As a result, the video quality delivered by the two local algorithms clearly improves with fragmentation and is nearly on par with the global algorithm when using four fragments. This is good news as it means that in certain cases by fragmenting the video segments to small enough pieces we can achieve almost optimal video quality using the locally optimal algorithms.

7.3 Longer Time Scale Changes in Available Bandwidth

So far we have only simulated such upload rate variation that happens in short time scale and the average rate remains constant. That kind of variation could be attributed to client churn in a base station and time dependent variations in channel quality due to, e.g., interference. So, the mobile device itself remains stationary. In this section, we investigate what happens when the average upload rate changes during the streaming session. Such variation happens in a longer time scale and would typically be caused by the uploading mobile device moving from good coverage area to a worse one causing degradation of the link quality.

We studied the above described scenarios by letting the upload rate vary around the mean as before but by changing the mean rate itself. We simulated two situations where within the time it takes to generate all the video segments (e.g, video duration), the average upload rate either linearly increases from 1 to 4 Mbps or vice versa, after which the average upload rate stabilizes to 2 Mbps in both cases. Hence, the overall average upload rate is the same over the whole session in the two scenarios.

The results on the left of Figure 11 show that when the average upload rate increases, the locally optimal algorithms are unable to anticipate the rate change in the way that the oracle is (knapsack global). They cannot match the video quality of the oracle algorithm even with fragmentation enabled. However, this effect is not visible in the scenario where the average upload rate decreases. In that case, the locally optimal algorithms with fragmentation provide a video quality that is on par with the oracle.

The reason for this difference in behavior can be explained by reasoning what is the advantage of the oracle in both cases. In case of increasing upload rate, the oracle algorithm can take more risky choices in uploading enhancement layer chunks of earlier segments even if it slows down the progress of uploading the lower layer chunks because it knows that later on it will have more capacity to catch up. The locally optimal algorithms do not know this and, therefore, they will choose to upload lower layers first that improve the PSNR the most. However, later on when the upload rate has increased and there is plenty of capacity left over, it is too late to upload the enhancement layers of the earlier segments. This phenomenon does not exist in the case of decreasing average upload rate.

To conclude, these results demonstrate that being able to predict the upload rate and to apply the global knapsack algorithm has potential to improve the results compared to the local knapsack or greedy algorithm, especially when the mean rate increases over a relatively long period of time.

8 Related Work

Layered video coding has been around for well over a decade, hence its use in a number of wireless environments has been extensively studied. For instance, Wu et al. [10] present one of the first pieces of work that studies the use of layered coding to transmit video over broadband wireless networks. Schierl et al. [11] discuss in a more general sense the use of scalable video coding with mobile video transmission. More recently, Migliorini et al. [12] analyze SVC video streaming over WiMax and look specifically at the impact of different encoding options. Van Der Schaar et al. [13] investigate how to optimize SVC video quality when transmitted over 802.11a/e that uses coordinated channel access (HCF) instead of random access MAC. SVC streaming in mobile ad-hoc networks has also been investigated [14], including scenarios with multiple video sources [15]. Hsu et al. [16] study broadcasting of scalable video streams over dedicated broadcast networks. Hu et al. [17] propose algorithms for femtocell cognitive radio networks that enable near optimal delivery of scalable video so that capacity use is maximized and interference limited. Kang et al. [18] consider how to schedule scalable video for 3G broadcast and multicast. They emphasize protecting the base layer with ARQ mechanisms to avoid missed segments. Other works focus on more specific scenarios or requirements. Ramzan et al. [19], for instance, present an overview of peer-to-peer (P2P) streaming of scalable video. Also Abboud et al. [20] studied the use of SVC in P2P Video-on-Demand systems. They specifically looked into the SVC layer selection algorithms and the quality metric trade-offs within. An interesting application of SVC is presented by Hsu et al. [21]. They leverage the scalability to trade quality to battery life of a mobile device when the latter is scarce. The use of caches in the context of scalable video has also been investigated. For example, Kangasharju et al. [22] study the problem of distributing layered video over the Internet through caches. Specifically, they examine which videos and which layers in the videos should be cached. They model their problem as a 2-resource stochastic knapsack and present heuristics to solve it.

Joint source channel coding has also been applied in the context of wireless transmission. Kondi et al. [23] present a system in which the video is encoded by accounting for the characteristics of the wireless channel over which the video will be transported. Specifically, they investigate the application of unequal error protection (UEP) to scalable video, making more efforts to correctly transport the most important frames. Others also focused on optimizing error control schemes for SVC. For instance, Kang et al. [24] propose a cross-layer error control scheme for scalable video broadcasting in CDMA2000 systems. Similarly, Zhang et al. [25] present a UEP scheme for scalable video transmission over 3G network. Also Yang et al. [26] propose to employ UEP with scalable video but they focus specifically on error differentiation, i.e., congestion vs. bit-error induced packet loss, on the transport level. Such loss differentiation is no longer typically done as modern mobile networks provide a highly reliable service, such as the LTE link layer that runs a two-layer ARQ mechanism in order to prevent any bit errors from propagating to higher layers [27]. More recently, Ji et al. [28] have also studied joint source-channel coding for broadcasting to heterogeneous devices, Li et al. [29] design a scheme that combines SVC and adaptive modulation and coding for wireless video multicasting, Lin et al. [30] also study video multicasting in a multirate wireless network proposing a rate scheduling model that optimizes the visual quality for a multicast group while guaranteeing a minimum quality to all clients, and Otwani et al. [31] develop optimal scheduling policies for scalable video transmission over single and multiuser wireless channels. Our work is complementary to theirs in that joint source-channel coding could be used to improve the efficiency of the transmission of individual segments over the wireless channel. However, merging the two solutions may be challenging: We do not know precisely when a video segment will be transmitted when using our scheduler. Therefore, it is difficult to tune encoding parameters since channel characteristics, e.g., bandwidth, may have changed by the time of transmission.

The possibilities of SVC in the context of DASH video streaming have received increasing attention in the last few years. The challenges and opportunities presented by the use of scalable video content in DASH are investigated, for instance, in [4], [32], which particularly focus on network caches. Zahran et al. [33] show how multiple TCP connections can be used on high-delay links in order to optimize the performance of DASH streaming of scalable video content. Probably the closest pieces of work that we are aware of are those presented by Andelin et al. [8] and Chen et al. [34]. They both investigate layer switching algorithms for clients receiving SVC content and, in the former, combined with DASH. Differently from our work, the authors focus on the downstream scheduling of stored SVC-encoded content. Wang et al. [35] also present an adaptive framework for streaming stored SVC content. Although the problem formulation in the work by Li et al. [36], which presents a solution for distributed streaming quality adaptation, resembles that of ours, their system is designed for streaming on-demand pre-encoded SVC content. Hu et al. [37] approach the video quality optimization problem in wireless networks with a proxy deployed at the network edge. The proxy delivers SVC streams at dynamically cho-

sen rates that are selected on the basis of the queue size at the access network. Freris et al. [38] studies delivery of on-demand scalable video to clients that are connected to multiple networks having different characteristics, such as cognitive multiradio devices. Also Xing et al. [39] have investigated a similar problem.

For the case of mobile video upload it is possible to use a server to transcode video and redistribute it using plain DASH [40], but a near real-time system can be achieved only by limiting the video resolution, otherwise the server becomes a bottleneck. This is a strong motivation for our approach to use a standard stateless HTTP servers and produce ready-to-distribute content directly on the mobile device. Our approach requires the mobile device to be capable of real-time encoding SVC video which has been shown to be feasible [41] but is currently not widely available.

Quality of experience aspects of video streaming have also been studied extensively. Recent examples include [42] that report on a large scale measurement study of mobile video usage. Krishnan et al. [43] study the effect of initial joining time and buffering events on the engagement in watching videos. Dobrian et al. [44] provide some insights into mapping the considered QoE metrics to user behavior through a utility function and present a predictive model of Internet video QoE in their follow-up work [45].

Our previous work [7] tackled the issue of SVC upload over HTTP from mobile devices to multiple clients by presenting some heuristic scheduling strategies. The present work substantially extends it with an analytical formulation and analysis of the problem considering both global and local optimization goals, as well as optimized algorithms (both optimal and approximate) that can run on mobile devices with limited complexity. The simulation results show that performance is close to the highest achievable by an oracle-based optimal scheduling algorithm.

The encoding workload by the mobile device increases since scalable coding adds complexity, which may raise concerns about the impact on battery life. It is clear that SVC encoding will increase the energy consumption compared to non-scalable encoding. However, hardware-accelerated encoding process in smartphones appears to be a relatively small contributor to the total energy consumption compared to the camera itself when shooting and streaming live video [46]. Moreover, Chen et al. demonstrated the feasibility to design hardware-supported SVC encoders with reasonable energy consumption (300-400 mW) [41].

9 Conclusions

This work investigated the problem of designing optimized adaptation strategies for a multi client scenario when a live video content is shared by a mobile user. Scalable video coding and chunked HTTP streaming have been considered, as well as stateless HTTP servers not to negatively influence the scalability of the system. The upload scheduling problem has been analytically formulated, its complexity has been analyzed and a globally optimal, a locally optimal, and a simple greedy algorithms have been provided. Simulation results show that in most cases the local and greedy algorithms perform equally well and that in some cases they can deliver nearly as good video quality as the globally optimal solution. However, we also demonstrate other situations where they miss opportunities to improve the video quality and perform inferior to the globally optimal algorithm. The complexity of the proposed algorithms is limited, hence they can be effectively employed in mobile devices.

Acknowledgments

This work was supported by the Academy of Finland, grant number 278207, and Aalto Science Institute.

References

[1] ISO/IEC 23009, "Dynamic adaptive streaming over HTTP (DASH)," 2012.

[2] J.-R. Ohm, "Advances in scalable video coding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 42–56, 2005.

[3] ISO/IEC 14496-10 & ITU-T H.264, "Annex G extension of the H.264/MPEG-4 AVC," Nov. 2007.

[4] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louedec, "iDASH: improved dynamic adaptive streaming over HTTP using scalable video coding," in *Proceedings of the 2nd ACM conference on Multimedia systems*, San Jose, CA, USA, Feb. 2011, pp. 257–264.

[5] G.J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, Nov. 1998.

[6] R. P. Stanley, *Enumerative Combinatorics*, vol. 1, Cambridge University Press, 2nd edition, Dec. 2011.

[7] M. Siekkinen, A. Barraja, J.K. Nurminen, and E. Masala, "Exploring the delay versus quality tradeoff in real-time streaming of scalable video from mobile devices," in *Proc. of 2nd IEEE Intl. Workshop on Mobile Multimedia Computing (MMC ICME Workshop)*, Torino, Italy, June 2015.

[8] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, "Quality selection for dynamic adaptive streaming over HTTP with scalable video coding," in *Proceedings of the 3rd ACM Multimedia Systems Conference*, Feb. 2012, pp. 149–154.

[9] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 3, pp. 24:1–24:19, July 2012.

[10] D. Wu, Y.T. Hou, and Y.-Q. Zhang, "Scalable video coding and transport over broadband wireless networks," *Proceedings of the IEEE*, vol. 89, no. 1, pp. 6–20, Jan. 2001.

[11] T. Schierl, T. Stockhammer, and T. Wiegand, "Mobile video transmission using scalable video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1204–1217, Sept. 2007.

[12] D. Migliorini, E. Mingozzi, and C. Vallati, "Performance evaluation of H.264/SVC video streaming over mobile WiMAX," *Computer Networks*, vol. 55, no. 15, pp. 3578–3591, 2011.

[13] M. Van Der Schaar, Y. Andreopoulos, and Z. Hu, "Optimized scalable video streaming over IEEE 802.11 a/e HCCA wireless networks under delay constraints," *IEEE Transactions on Mobile Computing*, vol. 5, no. 6, pp. 755–768, June 2006.

[14] M. Qin and R. Zimmermann, "An adaptive strategy for mobile ad hoc media streaming," *IEEE Transactions on Multimedia*, vol. 12, no. 4, pp. 317–329, June 2010.

[15] T. Schierl, K. Ganger, C. Hellge, T. Wiegand, and T. Stockhammer, "SVC-based multisource streaming for robust video transmission in mobile ad hoc networks," *IEEE Wireless Communications Magazine*, vol. 13, no. 5, pp. 96–103, Oct. 2006.

[16] C.-H. Hsu and M. Hefeeda, "Flexible broadcasting of scalable video streams to heterogeneous mobile devices," *IEEE Transactions on Mobile Computing*, vol. 10, no. 3, pp. 406–418, Mar. 2011.

[17] D. Hu and S. Mao, "On medium grain scalable video streaming over femtocell cognitive radio networks," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 3, pp. 641–651, Apr. 2012.

[18] K. Kyungtae, Y. Cho, J. Cho, and H. Shin, "Scheduling scalable multimedia streams for 3G cellular broadcast and multicast services," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 5, pp. 2655–2672, Sept. 2007.

[19] N. Ramzan, E. Quacchio, T. Zgaljic, S. Asioli, L. Celetto, E. Izquierdo, and F. Rovati, "Peer-to-peer streaming of scalable video in future Internet applications," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 128–135, Mar. 2011.

[20] O. Abboud, T. Zinner, K. Pussep, S. Al-Sabea, and R. Steinmetz, "On the impact of quality adaptation in SVC-based P2P video-on-demand systems," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, New York, NY, USA, 2011, MMSys '11, pp. 223–232, ACM.

[21] C.-H. Hsu and M. Hefeeda, "Achieving viewing time scalability in mobile video streaming using scalable video coding," in *Proc. of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys)*, Scottsdale, AZ, 2010, pp. 111–122, ACM.

[22] J. Kangasharju, F. Hartanto, M. Reisslein, and K.W. Ross, "Distributing layered encoded video through caches," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 622–636, 2002.

[23] L.P. Kondi, F. Ishtiaq, and A.K. Katsaggelos, "Joint source-channel coding for motion-compensated DCT-based SNR scalable video," *IEEE Transactions on Image Processing*, vol. 11, no. 9, pp. 1043–1052, Sept. 2002.

[24] K. Kyungtae and W.J. Jeon, "Differentiated protection of video layers to improve perceived quality," *IEEE Transactions on Mobile Computing*, vol. 11, no. 2, pp. 292–304, Feb. 2012.

[25] Q. Zhang, W. Zhu, and Y.-Q. Zhang, "Channel-adaptive resource allocation for scalable video transmission over 3G wireless network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 8, pp. 1049–1063, Aug. 2004.

[26] F. Yang, Q. Zhang, W. Zhu, and Y.-Q. Zhang, "End-to-end TCP-friendly streaming protocol and bit allocation for scalable video over wireless Internet," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 4, pp. 777–790, May 2004.

[27] A. Larmo, M. Lindstrom, M. Meyer, G. Pelletier, J. Torsner, and H. Wiemann, "The LTE link-layer design," *IEEE Communications Magazine*, vol. 47, no. 4, pp. 52–59, Apr. 2009.

[28] W. Ji, Z. Li, and Y. Chen, "Joint source-channel coding and optimization for layered video broadcasting to heterogeneous devices," *IEEE Transactions on Multimedia*, vol. 14, no. 2, pp. 443–455, Apr. 2012.

[29] P. Li, H. Zhang, B. Zhao, and S. Rangarajan, "Scalable video multicast with adaptive modulation and coding in broadband wireless data systems," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 57–68, Feb. 2012.

[30] K.C. Lin, W.-L. Shen, C.-C. Hsu, and C.-F. Chou, "Quality-differentiated video multicast in multirate wireless networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 1, pp. 21–34, Jan 2013.

[31] J. Otwani, A. Agarwal, and A.K. Jagannatham, "Optimal scalable video scheduling policies for real-time single- and multiuser wireless video networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 6, pp. 2424–2435, June 2015.

[32] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louedec, "Efficient HTTP-based streaming using scalable video coding," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 329–342, 2012.

[33] S. Ibrahim, A. H. Zahran, and M. H. Ismail, "SVC-DASH-M: Scalable video coding dynamic adaptive streaming over HTTP using multiple connections," in *IEEE 21st International Conference on Telecommunications (ICT)*, May 2014, pp. 400–404.

[34] S. Chen, J. Yang, Y. Ran, and E. Yang, "Adaptive layer switching algorithm based on buffer underflow probability for scalable video streaming over wireless networks," *IEEE Transactions on Circuits and Systems for Video Technology*, 2015.

[35] X. Wang, M. Chen, T.T. Kwon, L.T. Yang, and V.C.M. Leung, "AMES-cloud: A framework of adaptive mobile video streaming and efficient social video sharing in the clouds," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 811–820, June 2013.

[36] X. Li and B. Veeravalli, "A differentiated quality adaptation approach for scalable streaming services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, pp. 2089–2099, Aug. 2015.

[37] H. Hu, X. Zhu, Y. Wang, R. Pan, J. Zhu, and F. Bonomi, "Proxy-based multi-stream scalable video adaptation over wireless networks using subjective quality and rate models," *IEEE Transactions on Multimedia*, vol. 15, no. 7, pp. 1638–1652, Nov. 2013.

[38] N.M. Freris, C.-H. Hsu, J.P. Singh, and X. Zhu, "Distortion-aware scalable video streaming to multinetwork clients," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 469–481, Apr. 2013.

[39] M. Xing, S. Xiang, and L. Cai, "A real-time adaptive algorithm for video streaming over multiple wireless access networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 795–805, Apr. 2014.

[40] B. Seo, W. Cui, and R. Zimmermann, "An experimental study of video uploading from mobile devices with HTTP streaming," in *Proceedings of the 3rd ACM Multimedia Systems Conference*, Feb. 2012, pp. 215–225.

[41] L.-G. Chen, T.-D. Chuang, Y.-J. Chen, C.-T. Li, C.-J. Hsu, S.-Y. Chien, and L.-G. Chen, "An H.264/AVC scalable extension and high profile HDTV 1080p encoder chip," in *VLSI Circuits, 2008 IEEE Symposium on*, June 2008, pp. 104–105.

[42] M.Z. Shafiq, , J. Erman, L. Ji, A.X. Liu, J. Pang, and J. Wang, "Understanding the impact of network dynamics on mobile video user engagement," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Austin, TX, 2014, pp. 367–379.

[43] S.S. Krishnan and R.K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proc. of the 2012 ACM Conference on Internet Measurement Conference (IMC)*, Boston, MA, 2012, pp. 211–224.

[44] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," in *Proc. of the ACM SIGCOMM Conference*, Toronto, ON, Canada, 2011, pp. 362–373.

[45] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," in *Proc. of the ACM SIGCOMM Conference*, Hong Kong, China, 2013, pp. 339–350.

[46] S.V. Rajaraman, M. Siekkinen, and M.A. Hoque, "Energy consumption anatomy of live video streaming from a smartphone," in *Personal, Indoor, and Mobile Radio Communication (PIMRC), IEEE 25th Annual International Symposium on*, Sept 2014, pp. 2013–2017.

Matti Siekkinen obtained the degree of M.Sc. in computer science from Helsinki University of Technology in 2003 and Ph.D from Eurecom / University of Nice Sophia-Antipolis in 2006. He is currently a postdoctoral research fellow at Aalto University. His current research focuses on mobile computing and networking with a special interest in mobile multimedia services.

Enrico Masala received the Ph.D. degree in computer engineering from the Politecnico di Torino in 2004. In 2003, he was a visiting researcher at the Signal Compression Laboratory of UCSB. Since 2011 he is assistant professor in the Control and Computer Engineering Department at the Politecnico di Torino. His main research interests include performance optimization of multimedia communications over packet networks with particular emphasis on content distribution over the web.

Jukka K. Nurminen is a principal scientist at VTT and adjunct professor at Aalto University. In 2011-15 he spent five years as a professor at Aalto working on mobile cloud computing and energy-efficiency. He has almost 25 years experience of software research at Nokia Research Center.Jukka received his M.Sc degree in 1986 and Ph.D. degree in 2003 from Helsinki University of Technology. His research interest are focused on systems and solution that are energy-efficient, distributed, or mobile.