EXPLORING THE DELAY VERSUS QUALITY TRADEOFF IN REAL-TIME STREAMING OF SCALABLE VIDEO FROM MOBILE DEVICES

Matti Siekkinen[1]*, Alberto Barraja*[1,2]*, Jukka K. Nurminen*[1]*, Enrico Masala*[2]

[1] Department of Computer Science, School of Science, Aalto University, Finland
[2] Control and Computer Engineering Department, Politecnico di Torino, Torino, Italy
matti.siekkinen@aalto.fi, alberto.barraja@polito.it, jukka.k.nurminen@aalto.fi, masala@polito.it

ABSTRACT

This work addresses the increasingly common case of a user who wants to record a video with a mobile device and share it in (near) real-time with other users in the Internet using HTTP streaming. The proposed system builds on scalable video coding (SVC) to provide some degrees of adaptation to the users watching the video without the support of a costly high-performance low-latency video transcoding server. At the same time, the DASH standard is used to leverage the scalable video for providing HTTP streaming adaptation to users watching the video. The performance of several chunk uploading strategies are investigated by simulations, evaluating the tradeoff between several parameters such as video quality, buffering rate, idle time, and the startup delay of the clients. Finally, experimental results on a testbed confirm the simulation results.

Index Terms— scalable video coding, DASH, mobile video streaming, video upload, near real-time

1. INTRODUCTION

Recent years have witnessed a strong increase in mobile devices such as smartphones and tablets with powerful video acquisition capabilities. This allows real-time or near-real-time mobile video capturing and sharing to become a reality. However, the high variability of the wireless channel conditions to upload the content makes such operation rather challenging. Additionally, the only ubiquitously supported protocol for communications, HTTP, is notoriously ill-suited for multimedia communications. Fortunately, new video communications standards based on HTTP, such as the Dynamic Adaptive Streaming over HTTP (DASH) [1], can partially address this issue. The DASH media player can adapt to the various network conditions by appropriately choosing among the multiple quality levels available on the server.

In this context, this work focuses on a scenario where a user wants to share a live video captured by the mobile device through a standard HTTP web server. This could be the case, for instance, of a user sharing a live event with friends to allow them watching it in real-time. We assume that different viewers have different preferences: some prefer very short delay, while others appreciate high video quality.

We assume that the video is captured and encoded only at the mobile device. No additional transcoding would be done on the server side, which allows standard HTTP web servers to be used for cost-effectiveness. In this context scalable video coding [2] is a reasonable choice allowing different viewers consume videos with different qualities. At the same time, encoding all the different versions of the video in a format suitable for distribution using the DASH standard is desirable since it does not require the server to perform processing or adaptation that, instead, is directly performed by the DASH clients according to their adaptation logic. One of the most difficult challenges in the described scenario is to be able to optimally choose, at each time instant, which segment and layer should be uploaded to the server, given the current channel conditions.

Other works addressed some of these issues. For instance, the effectiveness of using SVC content for DASH video streaming in analyzed in [3, 4]. Ibrahim et al. [5] show how it is possible to optimize the performance of DASH streaming of SVC content on high-delay links by using multiple TCP connections. For mobile video uploading Seo et al. [6] present a system that can send video from a mobile device to a server that performs transcoding and adaptation for redistribution using DASH. Their work shows that a near real-time system can be achieved only if the video resolution is limited, otherwise the computational burden on the server is a bottleneck. Andelin et al. [7] is the study closest to our work, investigating layer selection algorithms for clients receiving DASH streaming of SVC content. However, unlike our work, it focuses on video viewing assuming that the already encoded content in SVC format is available in full on the server.

This work investigates the use of SVC and DASH (section 2) for sharing videos captured with mobile devices. We propose an SVC-based streaming system (section 3) and use it to study alternative policies for segment and layer selection. In particular, we analyze the tradeoff between video quality

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841

and delay through simulations (sections 4 and 5) and through measurements with an experimental testbed (section 6).

2. SCALABLE VIDEO CODING FOR DASH

A scalable video encoder allows to create an embedded bitstream that supports different quality-rate points. The embedded bitstream is implemented by a layered structure containing a base layer, which corresponds to the lowest supported quality and which can be decoded independently of the other layers, and additional enhancement layers, which improve the quality of the video but increase the required rate.

Different techniques exist to implement video scalability, the most important ones being spatial, temporal and quality (also named SNR) scalability [2]. In this work we focus on the scalable extension of H.264/AVC, Scalable Video Coding (SVC) [8], which supports all the above types of scalability. For simplicity, we focus only on spatial scalability, but the ideas underlying our approach can be extended to the other two forms.

The main advantage of layered coding is that the embedded bitstreams can be easily tailored to match the transmission resources, in particular the available bandwidth, by just selecting some layers and dropping others, without the need of transcoding or re-encoding the content.

The support for adaptation can be effectively exploited by standards such as DASH. The MPEG DASH [1] standardizes a popular approach for delivering multimedia content over Internet using HTTP-based streaming. DASH is adaptive and allows multiple quality levels to be made available to the client. Moreover, the specification is flexible enough to accommodate the support for describing and segmenting resources encoded using a scalable codec.

3. PROPOSED SVC-BASED STREAMING SYSTEM

3.1. Overview

The proposed system works so that a mobile node captures a video in real-time. When the upload process begin, a suitable Media Presentation Description (MPD) is prepared and uploaded by the mobile node to the server. The MPD description is parametrized so that there is no need to update it once every new resource becomes available on the server.

We would like to stress that, to achieve low latency, the client already encodes chunks with a predetermined fixed duration in time so that they can be made immediately available to the DASH clients without further processing in the server. This allows the mobile user to rely on any simple web hosting service without any additional logic for multimedia processing, keeping the cost of the system low but at the same time making the system scalable in terms of number of users that can connect.

One of the challenges in implementing such a system is to optimally tune the configuration of all subsystems so that the latency is minimized but the overhead does not significantly impact on the performance. An important issue is how to handle the bandwidth fluctuations that are typical of a wireless channel, which is the main focus of this work. Other issues include choosing the right DASH segment size which must be small enough so that frequent scheduling decisions can be taken by the uploader to minimize client latency, whereas too short segments must be avoided due to the inefficiencies of both the HTTP protocol and the coding process.

3.2. Uploading Strategies

As mentioned above, one of key elements of the system is the scheduler that decides, at each upload opportunity, which video segment should be uploaded. In fact, due to the variability of the channel bandwidth, it is not possible to proceed with a simple sequential upload of the various layers of the same segment otherwise the envisaged near real-time service, that keeps latency limited for users that do not want to wait for a better quality, cannot be achieved.

We explore the behavior of various different uploading strategies that we divide into two groups: naive strategies and adaptive strategies. The set of naive strategies upload the chunks corresponding to the different layers of video segments in a predefined order regardless of how the throughput fluctuates. If the next chunk scheduled for transmission has not yet been generated, the uploader blocks until it is available.

The most basic naive strategies are *horizontal* and *vertical* strategies. The horizontal one uploads all the base layer segments in order after which is continues by uploading the segments of the first enhancement layer in order and so on. Therefore, it is characterized by its nature to minimize delay before enhancing quality. In contrast, the vertical strategy prioritizes quality before delay by uploading each layer of a given video segment before moving on to upload the following segment. In addition, we define *diagonal* strategies that are characterized with a so called *steepness* parameter. These strategies upload video segments so that the lower the layer, the more chunks of that layer will have been uploaded at a given point of time. The steepness parameter determines the number of segments that a higher layer lags behind its immediate lower layer. Its value is between zero and one, zero being equivalent to the horizontal and one equivalent to the vertical strategy. Pseudocode corresponding to these strategies is presented in Algorithm 1.

Adaptive strategies, on the other hand, attempt to provide minimal number of buffering events for all clients. Hence, they strive to always deliver immediately the base layer chunk of a newly generated video segment. This is accomplished by adapting to bandwidth fluctuations: The uploader continuously monitors the throughput and uploads a higher layer

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841

Algorithm 1 Naive strategies

input: most recently generated segment number: $n_{last} = 1$;
input: current segment and layer numbers: $n_{cur} = 1; l = 0$;
input: steepness $s \in [0, 1]$;
repeat
  $upload\_chunk(l, n_{last})$;
  for all $l \in$ enhancement layers **do**
    $n_{cur} = get\_first\_segment\_not\_yet\_uploaded(l)$;
    $n_{prev} = get\_last\_segment\_uploaded(l-1)$;
    while $(n_{cur} < n_{prev} \times s)$ **do**
      $upload\_chunk(l, n_{cur})$;
      $n_{cur} = n_{cur} + 1$;
    end while
  end for
  $n_{last} = n_{last} + 1$;
  if needed, wait until segment n_{last} is available;
until stream ended and all chunks uploaded

Algorithm 2 Adaptive strategies

input: most recently generated segment number: $n_{last} = 1$;
input: current segment and layer numbers: $n_{cur} = 1; l = 0$;
input: time remaining until new segment: t_{rem} =segment length;
input: estimated upload time: $t_{est} = 0$;
input: measured upload rate and time: $r_{ul} = 0; t_{ul} = 0$;
repeat
  $t_{ul}, r_{ul} = upload\_chunk(l, n_{cur})$;
  $t_{rem} = t_{rem} - t_{ul}$;
  if (using gradual strategy) **then**
    if (all layer l chunks uploaded) **then**
      $l = l + 1$;
    end if
  else if (using moderate strategy) **then**
    $l = (l + 1) mod$ number of layers;
  else if (using steep strategy) **then**
    $l = get\_lowest\_layer\_with\_fewest\_uploaded\_chunks()$
  end if
  $n_{cur} = get\_first\_segment\_not\_yet\_uploaded(l)$;
  $t_{est} = predict\_next\_upload\_duration(l, n_{cur}, r_{ul})$;
  if $(t_{est} > t_{rem})$ OR (all chunks till n_{last} uploaded) **then**
    wait until segment $n_{last} = n_{last} + 1$ is available;
    $l = 0$;
  end if
until stream ended and all chunks uploaded

chunk only if it estimates that there is enough time to do that before the next video segment becomes available in order not to jeopardize the real-time delivery of the base layer. The estimate is computed based on the measured average upload rate during the previous chunk upload. We define three different adaptive strategies based on how they prioritize the delivery of different layers: *gradual*, *moderate*, and *steep*. Their names reflect how much the strategy allows higher layers to lag behind the lower layers, similarly to the naive strategies. Pseudocode corresponding to these strategies is presented in Algorithm 2. Figure 1 visualizes the way they progress with the uploading. After the period of low bandwidth when it again increases 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 enhancement layer 2. 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.

4. SIMULATION SETUP

In order to quantify the delay vs. quality tradeoff in different scenarios, we simulated the behavior of the different uploading strategies. We used real SVC-encoded video sequence information as input to the simulations. In particular, we encoded the standard video sequences known as *ice*, *crew*, *city*, *harbour* using, for the highest quality layer, 30 frames per second (fps) and 4CIF (704×576) resolution. We believe these sequences may well represent content that is live captured and transmitted to make it immediately available to viewers. We employed spatial scalability with three layers (base and two enhancements). The sequences are balanced in terms of spatial details, since *ice* 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 spatial enhancement layers. The layers have been encoded with a fixed quantization parameter in order to achieve approximately 100 kbit/s, 350 kbit/s and 900 kbit/s for each of the three layers, respectively.

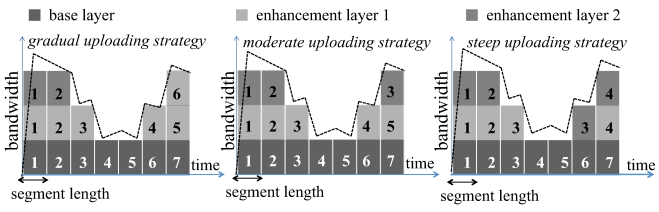


Fig. 1. Chunk uploading order for the different strategies.

The standard video sequences are short, therefore to generate a sufficiently long video sequence we concatenated the four sequences twice (*ice*, *crew*, *city*, *harbour*, *ice*, *crew*, *city*, *harbour*), which yields a 64 s video. Table 1 reports the encoding PSNR for the various layers of each sequence. We employed the first 240 frames of each sequence as this is the

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841

Table 1. Encoding PSNR (dB) when an increasing number of spatial layers is available.

Sequence	Layer 1	Layer 2	Layer 3
ice	29.69	33.90	38.93
crew	28.36	31.29	33.86
city	24.81	27.40	32.88
harbour	22.32	26.05	29.11

size of the shortest sequence in the set (*ice*).

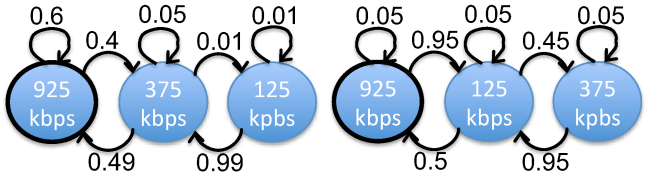
We experimented with two different channel bandwidth profiles, generated by means of a three-state Markov chain shown in Figure 2. Such models allow capturing the graceful degradation and improvement of the channel capacity while the user is mobile and of the available bandwidth when more clients join to share the resources of the wireless network that the user is connected to. The video is encoded in such a way that the average bitrate of all the layers combined together is slightly below the highest bandwidth level, while the average bitrate of the base layer is slightly above the lowest bandwidth level. Thus, we assume that the client has some knowledge of how much the bandwidth typically varies when choosing the encoding parameters. Hence, the absolute bandwidth values used here are not as important as their relative values to the bitrates of the video layers.

5. SIMULATION RESULTS

We study the behavior of different uploading strategies from several perspectives. First, we look at the performance tradeoffs with naive strategies. After that, we investigate what can be achieved through strategies that adapt to bandwidth variations. We also look at the impact of tuning the adaptation parameters. Finally, we examine the impact of splitting the video chunks into smaller pieces for transmission. All the results are evaluated as a function of different startup delays, which can be considered the willingness of the client to wait (i.e., be farther from real-time) in order to get better quality.

5.1. Naive Strategies

We summarize the different performance metrics in Table 2. The numbers indicate averages over a range of startup delays starting from zero and going up to 100 s. The buffering ratio



(a) good network conditions (b) poor network conditions

Fig. 2. Markov chain models for modeling wireless channel bandwidth fluctuations of a mobile access network.

Table 2. Main performance metrics for different non-adaptive strategies averaged over a startup delay range of 0-100 s.

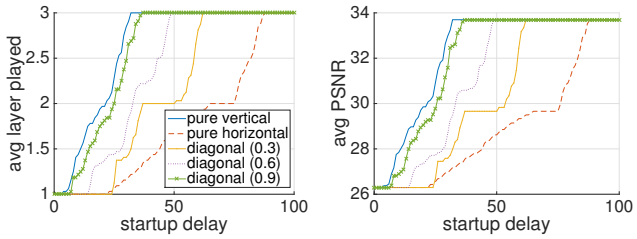
strategy	good network			poor network		
	layer	buf ratio	idle ratio	layer	buf ratio	idle ratio
vertical	2.61	0.07	0.03	2.02	0.53	0.03
horizontal	1.75	0	0.79	1.49	0	0.51
diagonal (s=0.3)	2.12	0.01	0.40	1.80	0.04	0.20
diagonal (s=0.6)	2.33	0.03	0.24	1.88	0.21	0.12
diagonal (s=0.9)	2.54	0.05	0.10	1.99	0.40	0.06

(*buf ratio*) is the time interval for which data is missing at the client (*buffering event*) due to unlucky scheduling decisions and the *idle ratio* is the amount of time in which the upload strategy suggests to wait. Both are normalized over the whole sequence duration. The horizontal strategy is clearly overly cautious leading to a large amount of time spent waiting for the next base layer segment. Conversely, the vertical strategy leads to a high average playback quality but suffers from relatively large amount of buffering time. The effects are emphasized when the network conditions are poor. The diagonal strategies seem to strike a tradeoff between these two extreme strategies, which agrees with intuition.

Let us take a closer look at some of the performance metrics. Figure 3 visualizes the delay vs. quality tradeoff with non-adaptive strategies in terms of average layer played and average PSNR of the resulting video as a function of startup delay. We make two main observations. First, the PSNR quality indicator varies in the exact same manner as the average layer played. This is because there is little difference in the sizes of different segments of a given layer. Therefore, each chunk of a given layer carries roughly similar amount of information and makes a similar magnitude enhancement to the overall video quality. For this reason, we use the average layer played as a good indicator of the resulting video quality in the rest of the paper. Second, we confirm that the diagonal strategies indeed strike a tradeoff between the horizontal and vertical ones. Furthermore, the steeper the strategy, the closer the strategy is to the vertical one.

5.2. Avoiding Buffering with Bandwidth Adaptation

We now turn our attention to the results obtained with the bandwidth adaptive strategies. We summarize again the main performance metrics in Table 3 for the two adaptive strategies.



(a) avg layer played (b) avg PSNR

Fig. 3. Average video quality as a function of startup delay in good network conditions.

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841

Table 3. Main performance metrics for different adaptive strategies averaged over a startup delay range of 0-100 s.

strategy	good network			poor network		
	layer	buf ratio	idle ratio	layer	buf ratio	idle ratio
vertical	2.61	0.07	0.03	2.02	0.53	0.03
horizontal	1.75	0	0.79	1.49	0	0.51
gradual	2.52	0	0.18	2.07	0	0.16
moderate	2.52	0	0.18	1.92	0	0.18
steep	2.31	0	0.43	1.66	0	0.34

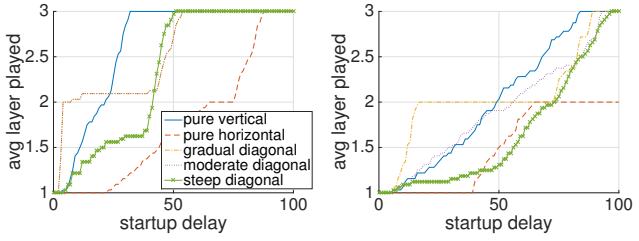
We list also here the vertical and horizontal strategies for ease of comparison. The bandwidth adaptive strategies, by design, reduce the buffering time to zero in all simulated scenarios, which indicates that any mistakes made in bandwidth estimation are not critical enough to lead to delay in delivery of a base layer segment.

Figure 4 plots the average quality vs. delay for the two different network conditions. We notice that the gradual strategy allows delivering the lower layers faster than the other strategies, while using the steep strategy in good network conditions makes it possible to play all three layers non-stop with a shorter startup delay than when using the other adaptive strategies but only by a small margin. In this scenario, the behavior of moderate strategy is exactly the same as the behavior of the gradual strategy and the curves completely overlap. In poor network conditions, the moderate strategy strikes a quality tradeoff between the steep and gradual ones. While the vertical strategy yields shortest delay in delivering all the three layers, it risks causing buffering events.

5.3. Relaxing the Real-time Requirements

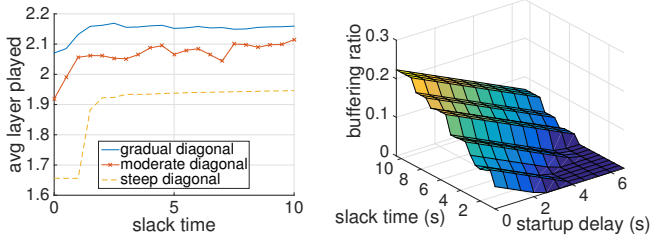
The default behavior of an adaptive strategy when making a decision concerning whether to upload a missing enhancement layer chunk is to compare the remaining time available until the next base layer chunk is available to the estimated upload time of the enhancement layer chunk under consideration. This behavior tries to ensure that each base layer chunk is delivered without any additional delay.

We wanted to check how much we could gain in terms of playback quality by relaxing this requirement. Hence, we



(a) good network conditions (b) poor network conditions

Fig. 4. Average layer played as a function of startup delay with bandwidth adaptive strategies.



(a) avg layer played (b) buffering ratio

Fig. 5. Impact of slack time parameter on the quality of experience. Average layer played is over a range of startup delays (0-100 s).

changed the default behavior by adding a parameter, called *slack time*, that is added to the remaining time before it is compared to the required upload time. Hence, the larger the *slack time*, the more delay we allow for the base layer delivery.

We plot the results in Figure 5(a) only with the poor network conditions, as the results with good network conditions are similar. At first, the average playback quality increases with the slack time but once the slack time grows beyond 3 s, there is no longer visible improvements. Note that here we have considered the overall playback quality for clients within the entire 0-100 s startup delay range. Hence, while a longer slack time value improves playback quality for clients that have a startup delay that is similar or longer than the slack time, it also worsens the quality for those clients having a shorter startup delay. The initial overall playback quality improvement with short slack time values is caused by reduced idle time for the uploading client.

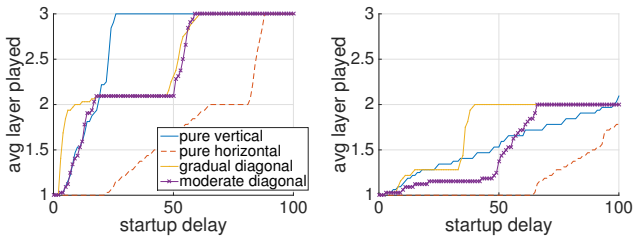
The effect of the slack time parameter is more striking on buffering ratio. Figure 5(b) shows how it changes with the slack time parameter for clients having different startup delays. We only plot one set of results since the impact on buffering ratio is similar for all the adaptive strategies. The shorter the startup delay of the client, the more it suffers from buffering events when the slack time parameter is increased. The clients with a startup delay at least 1.5 s longer than the slack parameter experience no buffering events.

5.4. Impact of video chunking

If the video is divided into large chunks, each chunk takes potentially a long time to deliver and the probability of the bandwidth to change during the upload is greater than in the case of video divided into small chunks. Furthermore, large chunks lead to potentially large amount of idle time because the uploader determines that it cannot upload the whole enhanced layer chunk before a new base layer chunk is available. Therefore, the larger the chunk, the less efficiently the available bandwidth is utilized by the adaptive uploader.

To quantify the above phenomenon, we simulated scenar-

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841



(a) good network conditions (b) poor network conditions

Fig. 6. Average layer played as a function of startup delay with testbed experiments.

ios where the 2 s video chunk is delivered in smaller fragments. Due to space constraints, we only mention the main takeaway: Delivering the video chunk in three fragments increases the average layer played by roughly 0.2 units at most (mean layer played of clients with 0-100 s startup delay) and dividing video chunks to even smaller fragments no longer improves the quality substantially. Such chunking could be easily achieved through HTTP-layer mechanisms, for example.

6. EXPERIMENTAL RESULTS

In order to understand whether our simulations capture all relevant phenomena, we also setup a testbed to perform experiments with the proposed techniques. The testbed is composed of a transmitter, running on a Linux PC, an emulated wireless link, a server that collects the uploaded HTTP segments, and clients that connect to the server to download the content. The wireless link was emulated by using the `tc` traffic shaper. The client uploads each video chunk by sending an HTTP POST requests on a web server. We implemented the vertical and horizontal strategies and gradual and moderate adaptive strategies.

We plot the results in Figure 6. The results match rather well those obtained through simulations when the network conditions are good. There are some discrepancies which are due to the fact that the client uses TCP whose throughput typically remains slightly below the available bandwidth and takes a few RTT rounds to ramp up, whereas in the simulations we assumed that the client can use all the bandwidth immediately from the beginning of a transfer of each chunk.

However, the results with the poor network conditions deviate substantially from the simulation results. The reason turns out to be TCP retransmissions which slow down the transmissions with the poor network conditions. The cause for the retransmissions is the relatively large changes in bandwidth that delay some packets so that the TCP retransmission timeout expires causing spurious retransmissions and drastic reduction in the TCP congestion window size. The deteriorating impact of such challenging network conditions on TCP performance is known to the community and various mechanisms to mitigate it have been proposed [9].

7. CONCLUSIONS

This paper presented a study of uploading SVC-encoded video chunks from a mobile device to a web server that makes them available for HTTP streaming using the DASH standard. We designed a number of different upload strategies using which we explored the inherent delay vs. quality tradeoff. The results show that by appropriately tuning the strategy, i.e. the prioritization between different layers, it is possible to optimize the quality given delay constraints for a number of different clients or, alternatively, optimize the delay given a set of quality constraints.

8. 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] 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.

[4] 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.

[5] 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)*, Lisbon, Portugal, May 2014, pp. 400–404.

[6] 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*, Chapel Hill, NC, USA, Feb. 2012, pp. 215–225.

[7] 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*, Chapel Hill, NC, USA, Feb. 2012, pp. 149–154.

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

[9] M. C. Chan and R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," *Wireless Networks*, vol. 11, no. 1-2, pp. 81–97, 2005.

The final publication is available at IEEE via: http://dx.doi.org/10.1109/ICMEW.2015.7169841