SVC-DASH-M: Scalable Video Coding Dynamic Adaptive Streaming Over HTTP Using Multiple Connections

Samar Ibrahim, Ahmed H. Zahran and Mahmoud H. Ismail
Department of Electronics and Electrical Communications,
Faculty of Engineering, Cairo University, Giza 12613, Egypt.
{ahzahran, mhismail}@ieee.org

***Abstract*—Dynamic adaptive streaming over HTTP (DASH) has gained a significant momentum for multimedia streaming due to its ability in crossing firewalls and availability of infrastructure. In the mean time scalable video coding (SVC) is building a similar momentum as it enables efficient media storage and caching. In this work, we identify the main components of adaptive SVC-DASH client and propose an streaming heuristic over dynamic multiple connections. The proposed algorithm is experimentally tested under different connection and link configurations. Our results show that the algorithm successfully achieves interruption free streaming under all the tested bandwidth and link configurations. Additionally, the usage of multiple connections results in noticeable improvements in the achieved streaming quality for large link delays.**

I. Introduction

Dynamic adaptive streaming over HTTP (DASH) [1] has gained a significant momentum and is widely adopted by many standardization bodies such as open IPTv and 3GPP. Generally, HTTP streaming has several attractive features that helped in speeding up its acceptance. Avoiding firewall and NAT traversal issues represents a major advantage for HTTP streaming over other traditional streaming techniques (e.g., RTP/UDP). Another advantage of using HTTP is the availability of existing HTTP cache infrastructures on the Internet, thus relieving not only the server load but also reducing the overall uplink traffic towards the cache. The main drawbacks of using HTTP streaming include higher end-to-end delays in the communication due to using transmission control protocol (TCP) and possible reduced link utilization due to TCP dynamics, but this can be compensated using the client-buffer.

Adaptive streaming techniques are commonly used to accommodate variations in the bandwidth to avoid annoying streaming interruptions due to buffer under-run. In HTTP streaming, the media is subdivided into short duration chunks, commonly known as *segments*. The granularity of streaming adaptation depends on the segment size and the adopted encoding technique. If advanced video coding (AVC) based on H.264/MPEG-4 Part 10 standard is used, the server would contain multiple versions of each segment encoded at different qualities (resolution, frame rates, picture quality or combination of them). Hence, in AVC-DASH, decisions are typically taken at the segment end. Scalable video coding (SVC) [2] represents an extension to AVC and it allows encoding a video segment into N layers including a *base* layer and numerous *enhancement* layers. The accumulation of more layers generally leads to improving the video quality. Hence, in SVC-DASH, the granularity of quality control extends both horizontally over time and vertically over layers.

This work is supported by the National Telecommunication Regulation Authority (NTRA) of Egypt

In this paper, we propose and evaluate the performance of an SVC-DASH client over multiple parallel connections (SVC-DASH-M) using a real testbed. In SVC-DASH-M, the quality of the requested layer-segment depends on the amount of buffered media and the network conditions. Additionally, it adopts a quality selection policy that is conservative for the frequency of video quality shifts. Our experimental results indicate that SVC-DASH-M successfully adapts to different operating conditions including various link delays and bandwidths. Our results also bring important insights regarding HTTP server configuration and HTTP performance.

The rest of the paper is organized as follows: in Section II, we present a brief overview for background and related work. The design of SVC-DASH-M is presented in Section III followed by its performance evaluation in Section IV. Finally, our conclusions and directions for future work are presented in Section V.

II. Background

In HTTP streaming, the content is requested using the HTTP GET request/response dialog that is typically serviced over TCP. Hence, HTTP streaming is considered a pull protocol in which the media is requested from the server according to the streaming client instructions. The details of the video information based on which the client takes the decision are typically obtained from a media presentation description (MPD) file that includes web links to the media files with corresponding encoding details of each file content.

Many studies focus on the design and performance evaluation of streaming over HTTP. These studies consider different types of video encoding (AVC[3] and SVC [4], [5]), different connection configurations (persistent versus non-persistent [6] and single versus multiple connections), different streaming modes (real-time versus stored) and various segment durations.

978-1-4799-5141-3/14/$31.00 ©2014 IEEE

Most of these studies focus on AVC-streaming over a single persistent connection.

More recently, several studies have shown that SVC-based streaming over HTTP has several merits and it started to get more attention. In [7], it is shown that SVC-DASH has several advantages including the possibility of serving a larger number of users with different equipment capabilities as well as having a higher caching efficiency. In [5], the authors assert that SVC-DASH not only needs less buffering requirements in comparison to AVC-DASH but it also improves the quality of experience (QoE) for the viewer. In [4], the authors compare the performance of different AVC-based and SVC-based heuristics using NS3 simulations with NS3 cradle. They also show that AVC performs better under high latencies while SVC better adapts to sudden and temporary bandwidth fluctuations when a single connection is used. The work in [8] analytically investigates the optimal selection policy for layer segment when SVC is used. The conducted simulations show that a vertical policy is optimal under fixed bandwidth while a diagonal selection policy is optimal for variable rate.

A few works considered using multiple connections in HTTP streaming. In [9], the authors present a rate-adaptive algorithm for AVC-based video streaming over two parallel connections. Their simulations show that parallel connections outperforms the serial segment fetching method in achievable media bit-rates but is slightly inferior in buffer underflow frequency. In [10], a cursor-based SVC client heuristic Live TV setting is considered where the client side buffer is limited. The authors also exploit using a fixed number of parallel connections and pipelined downloading of segment layers to overcome the high end-to-end delay issues. In our study, we investigate the usage of dynamic multiple connecitons for SVC-based streaming.

III. SVC-DASH-M Client Design

Typically, the first step performed by an HTTP streaming client is to download the MPD file and parse it to obtain the video information. For SVC-DASH, every video segment is split into multiple layers per segment depending on the encoding structure. The content of each layer segment is stored as a separate file on the HTTP server. Hence, the MPD information includes: segment ID, layer ID, layer-segment URL, layer-segment duration, and layer size. Note that the segment ID can be used to identify the timing information of the transmitted segment. Once the MPD information is received,SVC-DASH-M operation is based on three interacting components including a layer-segment requester, a connection manager, and enhancement layer selection policy.

Our layer-segment requester initially gets the first n_{min} base layer-segments, where n_{min} represents the minimum number of connections. Upon the reception of a complete layer-segment on connection c, SVC-DASH-M considers a level-based layer-segment requester that determines the quality of the requested layer-segment as shown in Algorithm 2. The details of reqLaySeg() is shown in Table I. The algorithm defines two application buffer-level thresholds denoted as B_{min} and B_{target}. B_{min} represents a low threshold for the

Algorithm 1 Level-based Layer-Segment Requester Algorithm.

```
bl = getBuffLevel(); //buffer level
 if bl <= B_min
 //request base layer only
    reqLaySeg(B,conn);
 elseif B_min < bl <= B_target
  //alternate base and enhancement
  //requests
   reqLaySeg(A,conn);
 else  // bl > B_Target
   //request enhancement layers only
    reqLaySeg(E,conn);
  end if
```

Algorithm 2 Connection Management Algorithm.

```
μ = SD/SFT;
if μ > 1
  r_nxt = estimateNextRate(lsType);
  ε = (r_nxt/r_prev);
  reqNextLaySeg(lsType, conn);
  if (n_c < n_c^max && μ > ε)
    reqNextLaySeg(lsType, -1);
elseif μ < 1
  if n_c > n_c^min
    close(conn);
  else
   reqNextLaySeg(lsType, conn);
  end if
end if
```

data to be maintained in the buffer to accommodate network condition variations while B_{target} represents a target buffer level that the application should be operating around. The defined policy in Algorithm 2 targets improving the streaming quality at high buffer levels through downloading enhancement layers. Additionally, the layer-segment requester policy maintains the user experience quality at low buffer-level through avoiding streaming interruptions by focusing on the base layer-segments.

The connection manager controls the dynamics of the used connections in SVC-DASH-M. In our algorithm, the number of opened connections, denoted as n_c, is lower-bounded by n_c^{min} and upper-bounded by n_c^{max}. The number of used connections varies when a layer-segment is received. Let c denoted the connection over which the layer-segment is received. Tne connection manager operation is shown in Algorithm 2. The details of the used functions are presented in Table I. The streaming performance is evaluated based on the segment duration-fetch ratio $\mu = \frac{SD}{SFT}$, where SD represents the duration of the received segment and SFT represents the duration over which that segment is fetched. Note that large values of μ indicate that the available bandwidth is large and vice verse. Hence, a new layer-segment is requested over c. An additional segment may be requested over a new connection if $n_c < n_c^{max}$ and there exists sufficient bandwidth for down-

`getBuffLevel( );`	returns the buffer level in sec.
`reqLaySeg(lsType, conn)`	requests a layer-segment from the server over connection conn. lsType could be "B" for base layer, "E" for enhancement layer, or "A" to alternate between base and enhancement layers.
`estimateNextRate(lsType)`	given the layer-segment type, this function returns the data rate required for downloading the next *two* segments to be requested.
`reqNextLaySeg(lsType, conn);`	given the layer-segment type, this function requests the next layer-segment to be requested over conn. If conn is set to -1, then a new connection is opened to request the selected layer-segment.

Table I: Algorithm Functions.

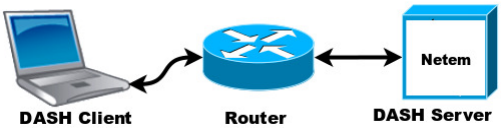

Figure 1: Testbed Architecture.

loading two segments. The bandwidth sufficiency condition is satisfied if the ratio between the required bandwidth for downloading the two selected layer-segments (denoted as r_{nxt}) to the estimated connection bandwidth (denoted as r_{prev}) is less than the segment duratoin-fetch ratio. To this end, it is worth noting that all the decision parameters are available or are readily measured at the application layer and do not incur any additional overhead to the communication process.

The third design component for SVC-DASH-M is layer-segment *selection policy*. Generally, this policy should ensure that any selected layer-segment is expected to be received before its playout time. In our client, this policy is implemented in *reqNextLaySeg(lsType, conn)*, which follows a horizontal selection policy by which no layer-segment from layer n is requested unless all the layer-segments of layer $n-1$ have been already downloaded. This policy aims to reduce the quality shifts in the received video. Other possible policies include vertical policy, which is greedy in nature as it downloads all enhancement layer-segments for segment i before downloading enhancement layer-segments for segment $i+1$. In between the two extremes, other policies may be developed. Irrespective of the chosen policy, estimating an accurate expected reception time for the selected layer-segment represents a challenging design issue because TCP performance depends on several factors including the link parameters (e.g., bandwidth, delay, and reliability) and TCP parameters (e.g., initial window size and congestion control algorithm). In our algorithm, we consider that a newly opened connection would equally share the estimated bandwidth of connection c. This design choice is motivated by the well-known friendly behavior of TCP. Investigating other selection policies and developing accurate SFT estimator is indeed worth investigating and is considered a future work.

IV. Performance Evaluation

A. Testbed Setup

Our evaluation is based on experimentation over a testbed shown in Figure 1 . In our evaluation, we used the forty-four-second Paris.yuv obtained from Arizona State University video library [11]. The video is processed using the Joint Scalable Video Model (JSVM) [12] as follows. First, the video is sliced into two-second segments using JSVM downsampler. Each segment is then encoded to ten scalable layers including spatial (original CIF and a downsampled QCIF), temporal and quality scalability. The encoder produces one .264 file per segment. We developed additional code to extract different portions corresponding to different layers in each segments using layer information that are obtained from JSVM bitstreamextractor. The corresponding MPD file is then compiled based on the extracted layer-segment information. Finally, the MPD file together with the layer segment files are uploaded to our HTTP server, a laptop running Ubuntu 12.04.

Traffic control tools [13] are employed to emulate different bandwidth and delay values. Netem is used to set the delay and delay standard deviation (assumed 10% of the delay). Additionally, token bucket filter (tbf) is used to set the link bandwidth to values of interest. For the proper setup of tbf, we set the $limit \geq Bw * HZ$ and $buffer(burst) = \frac{Bw}{HZ}$, where Bw represents the emulated bandwidth and HZ represents the Linux kernel timer interrupt frequency. Additionally, one may need to reset scatter-gather, tcp-segmentation-offload, generic-segmentation-offload and *generic-receive-offload* using ethtool to avoid packet dropping. Our client is implemented using C to connect and fetch the MPD file and video layer-segments according to the presented algorithm. The client assumes a playout latency of six seconds and a target level of ten seconds. If the buffer is depleted during the streaming, the application re-enters a buffering mode until it secures a six-second portion in the buffer again. Wireshark is used to trace the transmitted packets to estimate some performance metrics. Our key performance indicies (KPIs) include the number of interrupts (n_i), the number of opened connections (n_c), the number of closed connections due to low bandwidth (n_{cc}), the application downloaded data (d_v), the time at which the application stops downloading more layer-segments (t_d), the application goodput (γ), and the average quality (q). The application goodput here is estimated as the ratio between the total transmitted video layer-segment sizes and the time spent in transmitting this amount of data. These KPIs are evaluated for different link bandwidth and delay configurations.

B. Performance Results

Figure 2 plots the average received quality for the streamed video versus different bandwidth configurations for different min-max connection configurations and 10 ms link delay.

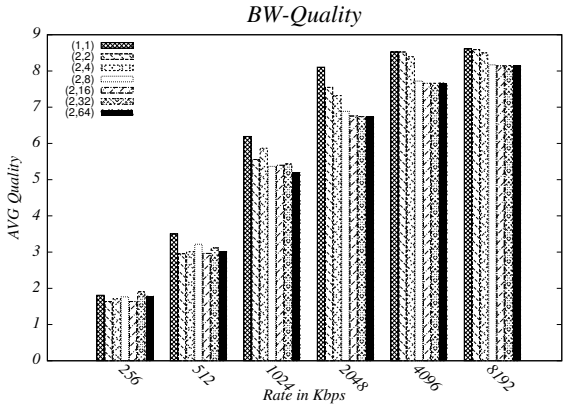

Figure 2: Segment Quality Versus Bandwidth for HTTP 1.1.

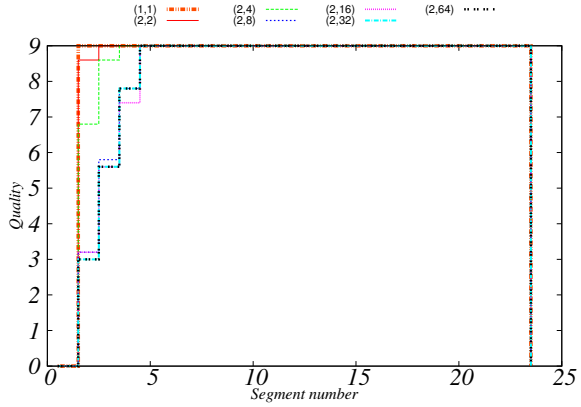

Figure 3: Segment Quality versus segment number for HTTP 1.1.

Note that the average quality is estimated over all received segments. Intuitively, increasing the bandwidth improves the quality of the received video for all connection configurations. Additionally, one can notice that the connection configuration has no significant impact on the received quality at a specific configuration. Furthermore, the figure shows that a single connection represents a good choice for the tested link configurations especially at low bandwidth values.

Figure 3 shows the achieved quality for each segment considering different connection configurations for an average link-delay of 10 ms and a bandwidth of 8 Mbps. The figure provides us with some insights on the quality dynamics versus time. The figure shows that all the connection configurations managed to download the highest quality towards the video session end. However, the main difference lies in the the starting dynamics. At the beginning of the streaming, the adopted conservative approach for layer-segment request implies requesting base layer-segments only and limits the benefit from the available bandwidth to download enhancement layer-segments. Note that this conservative approach helps reducing the initial playout latency, which is another important QoS metric. Once the minimum amount of media is secured in the buffer, the client starts to download enhancement layers for the subsequent segments. The figure also points out that the connection configuration has a direct impact on the streaming quality of the first few segments. Clearly, using a limited number of connections speeds the process of quality improvement at the beginning. This result can be interpreted by the

(n_c^{min}, n_c^{max})	(1, 1)	(2, 2)	(2, 4)	(2, 8)	(2, 16)	(2, 32)	(2, 64)
n_i	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n_c	1.00	3.00	13.80	18.60	17.20	19.40	18.40
n_{cc}	0.00	0.00	7.40	11.40	10.80	12.20	11.80
d_v (MB)	1.5	1.3	1.4	1.4	1.3	1.5	1.4
t_d (sec)	37.74	36.81	38.89	41.12	37.89	40.69	40.34
γ (kbps)	39.14	35.82	35.74	34.24	35.35	36.35	34.85
q	1.81	1.63	1.71	1.77	1.63	1.90	1.78

(a) 256 kbps Bandwidth.

(n_c^{min}, n_c^{max})	(1, 1)	(2, 2)	(2, 4)	(2, 8)	(2, 16)	(2, 32)	(2, 64)
n_i	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n_c	3.00	4.00	8.40	14.60	52.60	58.40	61.20
n_{cc}	0.00	0.00	0.40	1.20	1.00	2.80	1.40
d_v (MB)	7.1	7.1	7.0	6.8	6.8	6.8	6.8
t_d (sec)	6.53	6.54	6.45	6.10	6.17	6.12	6.12
γ (Mbps)	1.1	1.1	1.1	1.1	1.1	1.1	1.1
q	8.61	8.59	8.50	8.17	8.14	8.15	8.15

(b) Bandwidth 8 Mbps.

Table II: KPIs for Bandwidth 256 kbps and 8 Mbps.

TCP friendly behavior that splits the bandwidth among active connections. Hence, as the client opens more connections at the beginning, the bandwidth splitting leads to extended download time of every layer-segment. Consequently, such behavior prevents requesting higher quality layer-segments based on the adopted horizontal enhancement layer-segment request.

Table II shows some of the several KPIs for 256 kbps and 8 Mbps links with 10 ms link delay. As a general observation, the connection configuration has limited impact on all the estimated KPIs except for n_c and n_{cc}. For any configuration, the algorithm opens more connections as the upper-bound n_c^{max} increases. Typically opening these connections is associated with a signaling overhead for a connection-oriented protocol such as TCP. However, many of these connections are also closed by the algorithm as their segment fetch time is larger than the segment duration, especially at low bandwidth. As an example, approximately 12 connections out of 19 opened connection for (2,32) configuration are closed. This indicates that the upper bound of the connections should be carefully chosen to match the operating conditions. Another interesting observation is that n_c gets larger than n_c^{max} in some configurations especially those with small n_c^{max} and high bandwidth. For example with (1,1) configuration, three connections are opened instead of one connection. The reason for this strange behavior is that the default server configuration implies that no more than 100 objects should be transferred over the same persistent connection. Note that the number of communicated objects would typically scale with the number of encoded layers and video duration for a specific segment size. Our results also indicate that under all the tested configurations, the algorithm manages to adapt the streaming to different bandwidths and no interrupts are encountered. Also, as the link bandwidth increases, the application goodput (γ) increases. This increase is due to the ability to download more enhancement layer-segments and the typical drop of the download time (t_d). Consequently, a higher average quality is observed as the bandwidth increases.

Figure 4 plots the average video quality versus the average link delay for different connection configurations assuming

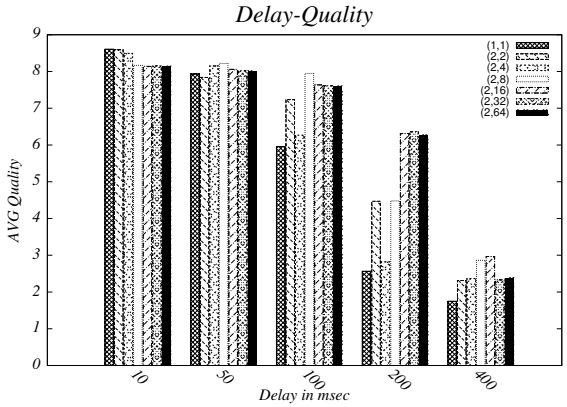

Figure 4: Quality versus Link Delay for HTTP 1.1.

(n_c^{min}, n_c^{max})	(1, 1)	(2, 2)	(2, 4)	(2, 8)	(2, 16)	(2, 32)	(2, 64)
n_i	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n_c	2.80	3.60	7.60	18.20	42.60	60.80	60.60
n_{cc}	0.00	0.00	0.00	0.80	0.00	2.80	2.80
d_v (MB)	6.7	6.6	6.7	6.9	6.7	6.7	6.7
t_d (sec)	16.89	10.04	6.87	6.85	7.00	7.25	7.24
γ (kbps)	399.69	662.30	979.10	1003.09	955.70	921.69	922.58
q	7.93	7.84	8.15	8.22	8.06	8.02	8.02

(a) Link Delay 50 ms.

(n_c^{min}, n_c^{max})	(1, 1)	(2, 2)	(2, 4)	(2, 8)	(2, 16)	(2, 32)	(2, 64)
n_i	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n_c	1.00	2.80	5.00	11.40	26.00	23.20	24.20
n_{cc}	0.00	0.00	0.80	2.00	12.60	11.20	12.60
d_v (MB)	1.3	1.5	1.5	2	2.2	1.5	1.5
t_d (sec)	32.51	29.99	19.91	22.63	30.26	21.00	21.33
γ (kbps)	38.99	50.77	74.25	87.91	73.18	70.77	70.91
q	1.75	2.31	2.37	2.86	2.97	2.33	2.39

(b) Link Delay 400 ms.

Table III: Key performance indicies for different link delays.

a bandwidth of 8 Mbps. It is important to note that we set the variance of the link delay to 10% of the average value. The results indicate that the quality significantly drops as the delay increases. A significant drop in the quality is noticed as the delay increases beyond 200 ms. The figure also indicates that using parallel connections pays off for large link delays and variance. The (2,8) configuration generally shows the best performance for different link delays. Table III shows the KPIs for 50 ms and 400 ms link delays. Similar to the previous results, the obtained results also show that the algorithm successfully streams the media without interruptions for different link delays. On the contrary, the results show noticeable variations for all the estimated KPIs for the same link configuration under different (n_c^{min}, n_c^{max}) values. The results also indicate that the number of closed connections due to limited bandwidth increases with the increase in delay for the configuration with high maximum connection bound. The application goodput also significantly deteriorates as the link delay increases. Additionally, the estimated goodput shows significant improvement when two connections are used in comparison to the serial download. Additional goodput gain is noticed between (2,2) and (2,4) for different link delays.

V. Conclusions and Future Work

SVC brings promising benefits to DASH including improved adaptive streaming and storage especially for highly heterogeneous networks. In this work, we presented the different key design components for SVC-DASH client considering multiple connections. Additionally, we presented SVC-DASH-M as a multiple-connection threshold-based client with horizontal layer-segment selection policy. Our experimental results show that using multiple connections has limited impact on the streaming performance for low link delays. On the contrary, it shows a noticeable improvement in the streamed quality for links with large delays. As a future work, we are interested in developing accurate estimators for layer-segment fetch time, investigating and developing new layer-segment selection policies, and developing criteria for determining the optimal connection configuration parameters.

References

[1] T. Stockhammer, "Dynamic adaptive streaming over http –: Standards and design principles," in *Proc. of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. New York, NY, USA: ACM, 2011, pp. 133–144.

[2] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[3] Xiaoling Qiu et. al., "Optimizing HTTP-based Adaptive Video Streaming for wireless access networks," in *Proc. 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, oct. 2010, pp. 838 –845.

[4] J. Famaey et. al, "On the merits of SVC-based HTTP Adaptive Streaming," in *in Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 419–426.

[5] R. Huysegems, B. D. Vleeschauwer, T. Wu, and W. V. Leekwijck, "SVC-Based HTTP Adaptive Streaming," *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 25–41, 2012.

[6] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over http dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 89–94.

[7] S. de la Fuente et. al., "iDASH: Improved Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding," in *in Proc. of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 257–264.

[8] Travis Andelin et. al., "Quality selection for dynamic adaptive streaming over http with scalable video coding," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 149–154.

[9] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj, "Rate adaptation for dynamic adaptive streaming over http in content distribution network," *Image Commun.*, vol. 27, no. 4, pp. 288–311, Apr. 2012.

[10] Niels Bouten et. al, "Minimizing the impact of delay on live SVC-based HTTP adaptive streaming services," in *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 1399–1404.

[11] [Online]. Available: http://trace.eas.asu.edu/yuv/

[12] H. Schwarz, M. Wien, and J. Vieron, "Joint scalable video model jsvm-5 software," Output Document from Joint Video Team, 2006.

[13] [Online]. Available: http://www.lartc.org/