SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP

Cong Wang and Amr Rizk*
University of Massachusetts Amherst
{cwang1,arizk}@umass.edu

Michael Zink
University of Massachusetts Amherst &
Technische Universität Darmstadt
zink@ecs.umass.edu

ABSTRACT

The dynamics of the application-layer based control loop of dynamic adaptive streaming over HTTP (DASH) make video bitrate selection for DASH a difficult problem. In this work we provide a DASH quality adaptation algorithm, named SQUAD, that is specifically tailored to provide high quality of experience (QoE). First, we review and provide new insights into why DASH rate estimation is hard. In a nutshell, in addition to the On-OFF behavior of DASH clients, there exists a discrepancy in the time scales that form the basis of the rate estimates across (*i*) different video segments, (*ii*) the rate control loops of DASH and TCP. With these observations in mind, we design SQUAD using with the objective of maximizing the average quality bitrate while minimizing the quality variations. We test our implementation of SQUAD together with a number of different quality adaptation algorithms of different methodologies under various conditions in the GENI testbed, as well as, in a cross-Atlantic measurement campaign. The measurement results show, that by sacrificing little to none in average quality bitrate SQUAD provides *significantly* better QoE in terms of quality switching and magnitude.

1. INTRODUCTION

Video Streaming is the most popular application in today's Internet. According to the latest Sandvine report [3] 67% of the downstream Internet traffic at peak hours in the US is real time entertainment such as live streaming and video on demand. Such high demand for video content requires approaches to efficiently transport the data from the content providers to the customer while providing the viewer a satisfying Quality of Experience (QoE).

To achieve this goal, virtually all video content providers have switched to the available bitrate streaming (ABR) mechanism. With ABR, the streaming rate, and thus the resulting quality of a video, can be adapted to the available bandwidth between the video server and the client. One of the most popular realizations of ABR streaming is the MPEG's Dynamic Streaming over HTTP (DASH) standard. Its popularity can mostly be attributed to the facts that *i)* DASH-format videos can be streamed from any kind of HTTP server; *ii)* adaptation logic resides in the client, which makes DASH highly scalable; and *iii)* it is an open standard. In addition to DASH, there exist proprietary ABR implementations like Microsoft's SmoothStreaming [4] , Apple's HTTP Live Streaming (HLS) [2], and Adobe's HDS [1].

*This work was funded in part by a DAAD exchange grant.

All of these ABR approaches use HTTP and the benefits of the underlying TCP protocol, which include standardized transport, firewall penetration, and adaptation to bandwidth changes. Unfortunately, the use of TCP also brings a set of difficulties which we identify and investigate in this paper. Examples of such challenges are, the dual control-loop (one for DASH and one for TCP as already identified by Huang et al. [13]), the impact of video segment size, and the impact of dead times on congestion window size.

An additional challenge for ABR video streaming is to provide a high QoE to the viewer. From the onset, DASH has been designed with the goal to prevent re-buffering events, which have the most serious impact on QoE, as shown by Krishnan and Sitaraman [17]. In addition to re-buffering events, frequent changes in quality have also been identified as detrimental to QoE by viewers, e.g., in Zink et al. [31].

These insights motivated us to design a new DASH adaptation algorithm that aims at achieving the highest possible quality (in terms of bitrate) while minimizing the number of quality changes, since we believe that most existing approaches only focus on the former and, thus, do not always result in a satisfying QoE. In this paper, we make the following contributions:

- **Critical Observations.** We perform a detailed analysis of the interplay between the DASH adaptation mechanism and underlying TCP. Through this analysis we identify several issues that contribute to the problem of optimizing the DASH streaming performance. Most notably, we identify *i)* the substantial impact of segment size on the download rate; *ii)* impact of dead time on the congestion window; and *iii)* the inaccuracy of segment-based available bandwidth estimation.

- **SQUAD.**We develop a new DASH rata adaption algorithm that has the goal to maximize the quality of experience of users watching video by addressing many of our critical observations. To achieve this goal, we consider two metrics that describe QoE and combine these in an online optimization algorithm.

- **Evaluation.** We present results from an extensive evaluation of the SQUAD algorithm. We perform a series of experiments in a controlled environment (GENI testbed) and in the wild (public Internet) and compare SQUAD with other, existing DASH algorithms.

The reminder of the paper is structured as follows. We describe the basic architecture and adaptation mechanisms in Sec. 2. In Sec. 3, we analyze the issues that arise from

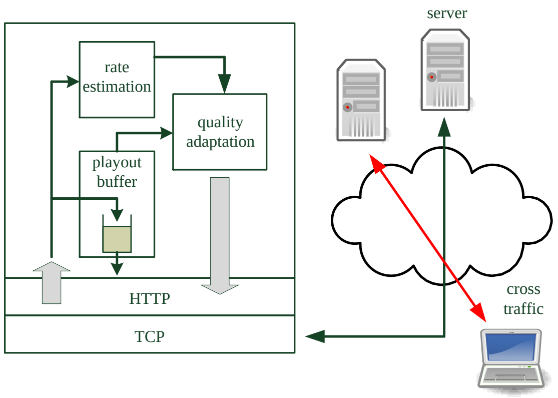


Figure 1: Coarse architecture of a DASH client. Buffer filling and download rate estimates are fed to the quality adaptation logic which decides on the quality of the next segment. The stream encounters varying network conditions, for example, due to contending cross traffic.

the interplay of DASH and TCP. Our SQUAD algorithm is introduced in Sec. 4 and a detailed evaluation of the algorithm is presented in Sec. 5. Section 6 presents related work in the area of DASH adaptation algorithms and we conclude the paper in Sec. 7.

2. DASH PLAYER ARCHITECTURE

In this section, we highlight the main components of a general DASH player architecture as depicted in Fig. 1. We divide the client block into the following logical components with specific functionality: *(i) playout buffer*, *(ii) rate estimation / prediction* and *(iii) quality adaptation*.

In the following, we will provide some definitions of important DASH parameters to lay the ground for subsequent modeling. We denote the size of a DASH video segment (in bits) of a certain quality as $s_{i,q}$, with the segment number $i \in \{1, \dots, N\}$ and the quality level $q \in \{1, \dots, Q\}$. Here, $q = 1$ ($q = Q$) denotes the lowest (highest) quality with respect to the video quality bitrate r_q, i.e., $r_j < r_{j+1}$ for $j \in \{1, Q-1\}$. We denote the quality of a fetched segment i by q_i and its quality bitrate by r_{q_i} and drop the subscript when it is obvious. We consider the case where all video segments have equal length in time, i.e., every segment carries X seconds. The fetch time of segment number i in quality level q is given as $t_{i,q} = s_{i,q}/R_{i,q}$, where $R_{i,q}$ is segment download rate in bit/sec. In Sect. 3 we describe how the segment download rate $R_{i,q}$ depends on a combination of multiple factors such as the network conditions, the TCP state and even the size of the segment $s_{i,q}$.

The playout buffer $B(i)$:. The purpose of the playout buffer is to smooth short-term variations of the network conditions. Specifically, *temporary* fluctuations in the segment download rates R should be absorbed by the buffer resulting in corresponding fluctuations in the buffer filling $B(i)$ while keeping the steady state playback quality q unchanged.

In this work, we define $B(i)$ as the cumulative number of video seconds contained in the playout buffer after fetching segment i. Note that $B(i)$ is defined over $i \in \{0, \dots, N\}$ and that by convention $B(0) = 0$. We define the event $\{B(i) = 0\}$ as the *rebuffering* event. For any practical application, the playout buffer is set to a finite size $B_{\max}$. For $B(i-1) < B_{\max}$ we write down the buffer recurrence as

$$B(i) = \max\{0, B(i-1) + X - t_i\}, \qquad (1)$$

where t_i is the fetch time of segment i. For $B(i) = B_{\max}$ we have $B(i+1) = B_{\max} - t_{i+1}$ since the player idles for X seconds when the buffer is full.

Rate estimation:. One basic client-side download rate estimation logic in DASH simply divides the segment size $s_{i,q}$ over the segment fetch time t_i. Here, the fetch time is given by the time difference between the timestamps of the HTTP `GET` request and the segment being delivered to the playout buffer, i.e.,

$$d_{i,q} = \frac{s_{i,q}}{t_i^{\texttt{delivered}} - t_i^{\texttt{GET}}} \qquad (2)$$

Note thate the rate estimate $d_{i,q}$ is smeared by the one-way delay of the `GET` request. This error diminishes with increasing segment size $s_{i,q}$. Rate estimation methods that calculate the download rate over multiple concatenated segments extend (2) to

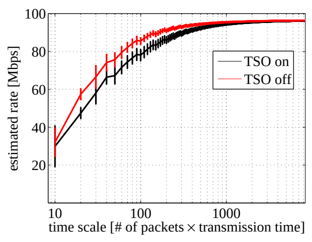
$$d_{i,j,q} = \frac{\sum_{k=i}^{j} s_{k,q}}{t_j^{\texttt{delivered}} - t_i^{\texttt{GET}}} \qquad (3)$$

for segment indexes $j \geq i$. In Sect. 3, we provide a critical evaluation of the foundations of the DASH rate estimates.

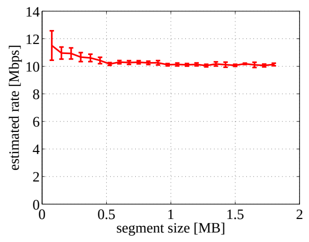
Quality adaptation:. Next, we review basic concepts for the segment based quality adaptation logic shown in Fig. 1. DASH clients first fetch media presentation description (MPD) files that contain information on the content to be streamed such as server IPs, bitrates of different quality levels and the URIs to the segments of different qualities [24]. Basically, quality adaptation algorithms have two sources of information, i.e., the status of the playout buffer filling and the download rate of previous segments provided by the rate estimation logic. Buffer based quality adaptation takes, in general, the current buffer filling $B(i)$ and in some cases also the change of the buffer filling $(B(j) - B(i)) \,/\, \left(t_j^{\texttt{delivered}} - t_i^{\texttt{delivered}}\right)$ for segments $j \geq i$, into account when deciding on the quality of the next segment to be fetched. The change of the buffer filling is an indicator for a mismatch of the segment download rate, which is an estimate of the available bandwidth, and the playout rate. In DASH, there exists a subtle difference between the change of buffer filling metric and the averaged download rate since DASH introduces inter-request time gaps. We will provide a detailed analysis of this issue in Sect. 3.

Quality adaptation mechanisms that additionally take the rate estimate into account aim to match the segment download rate to the playout rate by choosing segment qualities that have bitrates lower than the estimated available bandwidth. The estimation of the available bandwidth for the next segment(s) given the download rate history can rely, for example, on network active probing techniques [15] or simply on time series analysis. Further, the rate based adaptation component bears the risk that the chosen segment qualities oscillate in accordance to the available bandwidth. From a streaming point of view, it is well known that frequent video quality oscillations are detrimental to the quality of experience (QoE) perceived by the users.

In the following sections, we will refer to the basic DASH player architecture and describe the details of the modifications introduced by our adaptation algorithm SQUAD. In the next section, we provide some critical observations on DASH that sparked some of the specifications of SQUAD.



(a)



(b)

Figure 2: User space rate estimates in DASH: (a) Impact of NIC segmentation offloading on rate estimation in user space. (b) User space rate estimates in DASH become more accurate for longer segments.

3. CRITICAL OBSERVATIONS IN DASH

In the following, we review fundamental observations in Dynamic adaptive streaming over HTTP that make the problem of optimizing DASH performance a hard one.

3.1 DASH estimates are in user space

DASH clients *usually* estimate the segment download rate from segment timestamps in user space. Depending on the hardware environment and its configuration these estimates may vary significantly. Trivially, estimates made in virtualized environments may be highly varying, because of VM scheduling [28]. However, we first look at a non-virtualized bare-metal topology and show the impact of different settings of the network interface card (NIC), at the example of `segmentation offloading` (SO), on the DASH segment rate estimates. To this end, we use the Emulab testbed [27] to build a butterfly topology as depicted in Fig. 10 with $M = 1$ nodes on each side, where we use bare metal machines connected via 100 Mbps links. In this case we do not generate any cross traffic. In the first experiment, we emulate a persistent HTTP DASH flow through a long running greedy TCP flow and consider the segment download rate estimates obtained for different segment lengths. Figure 2a shows the rate estimates over increasing estimation time scales. We normalize the x-axis, i.e., the time scale, to the number of packets that fit into one time slot at line rate. From Fig. 2a we deduce that the user space rate estimate is indeed affected by the segmentation offloading for small DASH segments, i.e., in the order of 10's to few hundreds of packets. On the other hand, larger time scales, i.e., equivalently larger DASH segments, allow some averaging such that the impact of SO is negligible.

To get a better understanding of the DASH user space segment download rate estimates we rerun the experiment with the Python DASH player from [16]. We modified the player to continuously measure the download rate *during* segment download by taking one measurement point every 50 packets. A one-minute video is streamed in the topology in Fig. 10 with link capacities of 10Mbps, one DASH flow and no cross traffic. The video segment length is 2 sec. We measure the rate estimates for segments of different sizes as shown in Fig. 2b, which depicts averages and 0.95 confidence intervals. Here too, we observe that the inaccuracy of the rate estimates is highest when the segments are small.

3.2 DASH runs over TCP

In the following, we discuss specific characteristics of DASH that arise due to the fact that DASH utilizes TCP.

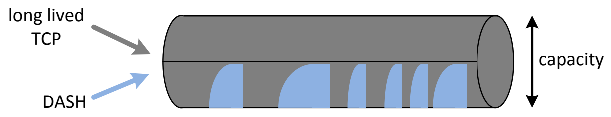


Figure 3: DASH abstraction as source of TCP mice flows.

Dual control loop

The design choice of DASH to utilize HTTP for adaptive bitrate streaming brings numerous advantages such as standardized transport, firewall penetration, adaptation to the bandwidth changes and all the advantages of TCP. However, since the DASH player needs to specify the quality bitrate of the segments to be fetched, it may be regarded as an outer control loop, while TCP running as an inner control loop that prevents congestion. While TCP aims for the fair share on a packet level time scale, DASH aims for the sustainable quality bitrate, i.e., essentially the fair share, on a segment level. The problem exacerbates since the control loop of DASH runs on a per segment basis. Since the segments are of different sizes, the time scale on which DASH tries to find the fairs share is continuously changing. Figure 4 shows empirical segment download rates for different segment sizes from testbed measurements with one DASH flow using the topology shown in Fig. 10 without cross traffic. We vary the segment lengths (in seconds), the link capacities and use DASH with persistent and non-persistent HTTP. Figure 4 clearly shows the impact of the DASH segment size on the download rate for both HTTP connection types. The empirical download rate may be much lower for small sized segments.

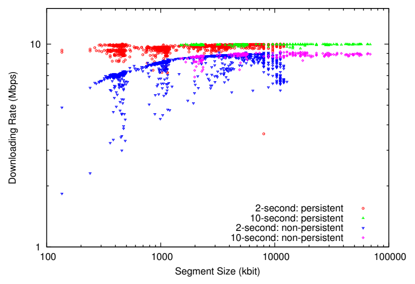
Our algorithm (SQUAD) which we present in Sect. 4 takes into account this discrepancy of the rate estimation time scales and provides DASH with (available) download rate information on the appropriate time scale.

DASH is TCP submissive

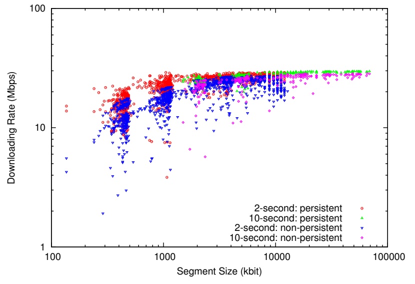
Although DASH utilizes HTTP over TCP/IP to retrieve segments, we argue that it does not necessarily receive its fair bandwidth share when competing with other long-lived TCP cross traffic. The reason for this behavior is that a DASH video stream does not constitute a *long-lived* TCP flow from the server to the client. Figure 3 shows a sketch of this behavior for one hypothetical case of one DASH flow competing with one long-lived TCP flow.

Reasons for this discontinuous traffic behavior lie in the nature of DASH streaming. First, assume a persistent HTTP connection, since in case of DASH over non-persistent HTTP it is simple to show that the continuous TCP cross traffic receives more than the fair bandwidth share. In general, there exist dead times of no DASH transmission that result from the DASH adaptation algorithm itself, i.e., depending on how often the DASH adaptation logic fetches a new segment and the corresponding buffer filling. Figure 5d shows the CDF of the dead times between receiving the last packet of one DASH segment and sending out the HTTP `GET` request for the next one. Clearly, there is silence time of multiple hundred milliseconds in the median case. This is sketched in Fig. 3 as gaps between the DASH segments.

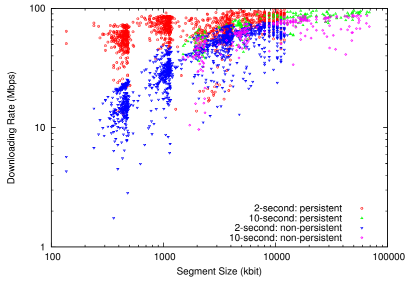
In Fig. 5 we show the results for the experiment of competing DASH and a long-lived TCP flow as sketched in Fig. 3.



(a) 10 Mbps

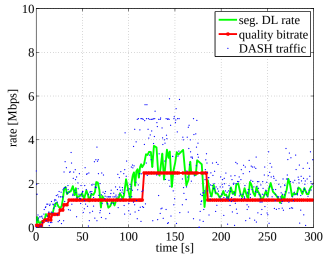


(b) 30 Mbps

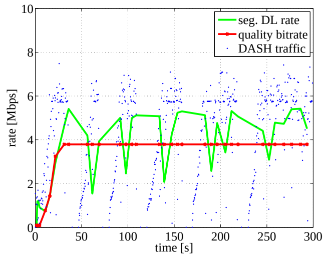


(c) 100 Mbps

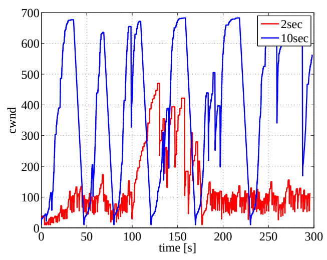
Figure 4: DASH segment download rates for links of different capacity. The segment size has a substantial impact.



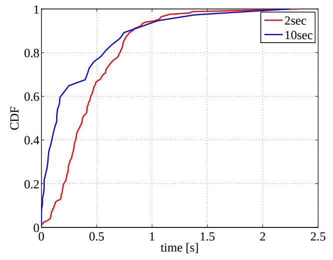
(a) DASH throughput 2sec.



(b) DASH throughput 10sec.



(c) DASH-TCP cwnd



(d) Inter-GET times

Figure 5: DASH is TCP submissive. Impact of the segment length for one DASH flow competing with one TCP Reno flow.

We run this experiment with two different segment sizes, i.e., 2 sec and 10 sec, a link capacity of 10 Mbps and a DASH repository of the `Big Buck Bunny` dataset with quality bitrates ranging from 89 Kbps to 4.2 Mbps. We emphasize that we do not aim to investigate ABR segment size optimizations in this work, but merely show that different settings of DASH parameters can lead to entirely different performance. Our aim is to show the fundamental roots for this varying behavior. Figure 5a shows the segment throughput as measured by the client as well as a finer grained throughput measurement that is done on the wire. For a segment size of 2 sec the quality bitrate of the DASH stream is remarkably low with respect to the fair share. The reason for that can be inferred from Fig. 5c, where we observe that the corresponding congestions window does not ramp up as the segments are "small" in size. In contrast, for 10 sec segments the congestion window in Fig. 5c is nearly an order of magnitude higher and the quality bitrate shown in Fig. 5b stays at the maximum available quality bitrate of 4.2 Mbps.

From a modeling perspective, the interrupted TCP stream generated by DASH can be regarded as the output of an ON-OFF source that generates TCP mice flows (although technically imprecise because of the persistent HTTP connection). Hence, it is known that competing long-lived (elephant) and mice flows suffer from fairness disparities [12].

4. SPECTRUM-BASED QUALITY ADAPTATION FOR DASH

In this section, we describe a DASH quality adaptation algorithm that addresses many of the critical points raised in Sect. 3. The aim of this algorithm is to maximize the quality of experience associated with DASH streaming session in a quantitative manner. We consider two metrics that describe QoE and connect these into an online optimization algorithm in a novel manner. Given a DASH streaming session of N segments each of length X seconds, the first metric we consider is the average video bitrate, i.e., the average video quality, which we express as

$$\bar{r} = \frac{1}{N}\sum_{i=1}^{N} r_{q_i}. \tag{4}$$

Recall that r_{q_i} is the quality bitrate of segment i at quality level $q \in \{1, \ldots, Q\}$. Formula (4) does not consider the detrimental impact of rebuffering on the average video quality. One method to capture rebuffering in (4) is to substitute N by N' which equals the number of segments N in addition to the number of segments that can be fitted in the rebuffering time as $\lceil t_{\text{rebuf}}/X \rceil$. Trivially, the corresponding rates r_{q_i} are set to zero.

The second metric that we consider is a centralized measure for the variation of the video quality around the average quality which is is denoted as "spectrum" in [31]. We adapt the spectrum definition to DASH streaming to express the video bitrate variation around the average bitrate given N segments as

$$H(N) = \sum_{i=1}^{N} z_i \left(r_{q_i} - \frac{\sum_{j=1}^{N} z_j r_{q_j}}{\sum_{j=1}^{N} z_j} \right)^2 \tag{5}$$

where $z_i = \mathbb{1}_{\{r_{q_i} \neq r_{q_{i-1}}\}}$. In the following, we describe our DASH quality adaptation algorithm that has the objective of maximizing (4) while minimizing the spectrum (5). From a classification point of view, the algorithm can be regarded as a rate **and** buffer based algorithm since it takes the information on the buffer filling as well as the historical download rates to decide on the quality of the next segment.

Smooth and reliable rate estimation

Given our observation in Sect. 3 that the measured segment download rate varies with the segment size and the current state of the server TCP state machine, we introduce two key ideas to provide smooth and reliable estimates of the download rate of the next segment to be fetched.

Smooth sub-segment rate estimates: Looking at the download rate estimation methods from Sect. 3 we observe

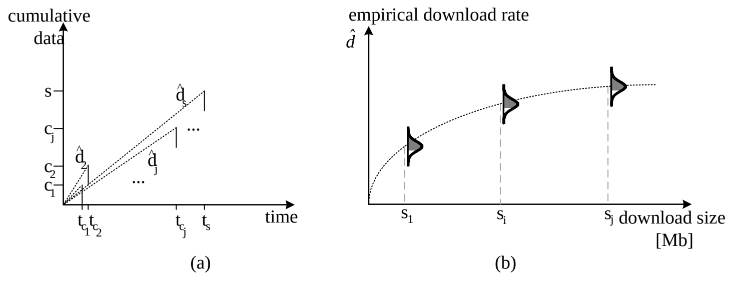


Figure 6: (a) Sub-segment rate estimates. (b) Empirical downloading rate vs. segment download size.

that the estimates are usually calculated upon the arrival of a video segment. Since segments vary in size, the different estimates correspond to different time scales. Hence, given the knowledge of the time scale dependence of the behavior of network protocols (such as TCP fairness) and thus the available bandwidth, we decide to take running estimates of the download rate vs. the downloaded data amount. Starting from the request time $t_n^{\texttt{GET}}$ for segment n we calculate the running download rate estimate as

$$\hat{d}_s = \frac{s}{t_n^{c_j,\texttt{delivered}} - t_n^{\texttt{GET}}} \tag{6}$$

where $t_n^{c_j,\texttt{delivered}}$ is the time required to download c_j bits of segment n in quality q, where $c_j \leq s_{n,q}$. Figure 6a depicts this procedure. In the following, we calculate $\hat{d}_s$ from (6) using the granularity of $c_{j+1} - c_j = 50$ IP packets, i.e., roughly 75kB.

Reliable rate estimates: One important contributor to the efficiency of any quality adaptation algorithm for DASH is the accuracy of the download rate estimate. However, in Sect. 3 we showed that the segment download rate in case of DASH depends on the segments size as well as the TCP state. Practical approaches to predict the TCP transmission rate that are based, e.g., on machine learning techniques have been considered in [25]. In contrast, in this work we assume a thin client that keeps track of the segment download history, however, we do not aim to predict the TCP transmission rate in such manner. Instead, we collect the download rates $\hat{d}_s$ similar to Fig. 4 and deduce an empirical lower bound on the download rate for the next segment $n+1$ of size $s_{n+1,q}$ from the percentile of the empirical distribution as

$$P\left[r_{n+1,q} \leq \hat{d}^{\varepsilon}_{s_{n+1,q}}\right] \leq \varepsilon, \tag{7}$$

where ε is a conservatism parameter. The smaller we set ε the more conservative is the lower bound $\hat{d}^{\varepsilon}_{s_{n+1,q}}$ for the download rate of the next segment. We sketch this procedure in Fig. 6b.

4.1 Init: A Slow Start of Segment Quality

Initially, we start the streaming session by retrieving the corresponding MPD file which we modified to include the segments sizes (in kB). Modifying the MPD to include segment size was first proposed by [16]. Since we do not assume any prior information on the available bandwidth along the path between the client and the streaming server we stream the first segment with the lowest possible quality. Instead of streaming only the first segment in the lowest quality we may stream a consecutive train of the first W_1 segments in the lowest quality. This conservative choice provides the possibility to estimate the download rate of higher quality segments that are usually larger in size. However, the larger W_1 the longer the player is stuck with the lowest quality, which increases the risk of video abandonment. In the sequel, we set $W_1 = 5$ if not stated otherwise.

The segment quality *slow start* behavior follows starting from segment $W_1 + 1$. Here, with every segment we double the quality level requested until we reach the highest possible quality. We conclude the initial phase and go to the next phase, that we denote the steady state, either when the quality slow start is finished or when the download time of the last fetched segment is longer than double the segment length X. We choose this empirical break condition to minimize the risk of rebuffering in the initial phase.

4.2 Steady state

In the following, we describe how the player decides on the quality of the next segment based on the algorithm we denote spectrum based quality adaptation.

4.2.1 Spectrum-based adaptation

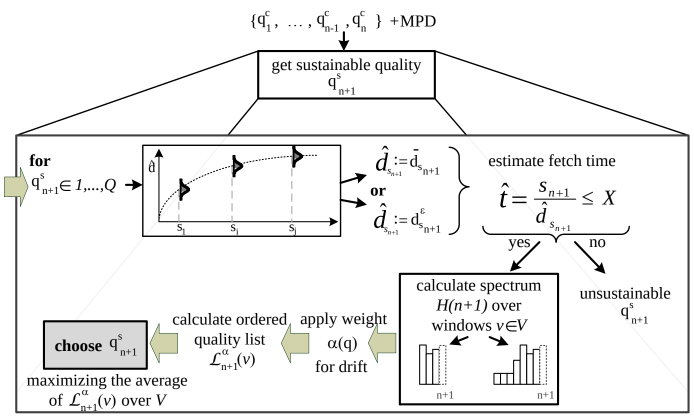
After the initial phase, i.e., $W_1 + \lfloor \log(Q) \rfloor$ segments, the player decides for the rest of the video stream on the segment qualities based on the spectrum H given in (5). Given the trace of quality levels downloaded so far, the player aims at minimizing (5) for all possible quality levels for the next segment. However, this strategy leads to preferring segment qualities near to the running average quality bitrate and may not necessarily have a drift towards higher qualities. Since we described the aim of this adaptation algorithm to maximize the average bitrate (4) subject to minimizing the variations around the mean (5) we modify the adaptation algorithm as follows.

First, we order the qualities according to their calculated spectrum H where we mark the qualities either as "sustainable" or "unsustainable" according to their estimated fetch time $\hat{t}$. *We denote a quality sustainable if its fetch time $\hat{t}$ is less or equal to the X seconds of video contained in a segment.* Considering the playout buffer evolution (1) this is equivalent to imposing a buffer constraint such that we do not fetch qualities that lead to buffer drain. In the sequel we will relax this constraint. This ordering operation is depicted in Fig. 7 and it provides the player with a first reference for choosing the quality of the next segment. We conjecture that, in general, movie consumers do not recall the movie quality after a few minutes such that the *current* QoE is function of the segment qualities over a window v of past segments. For example, it seems reasonable that during the course of a movie the viewers will less likely remember the quality of a scene after few minutes. Hence, as depicted in Fig. 7, given n downloaded segments with qualities q_i^C with $i \in \{1, \ldots, n\}$, we calculate the ordered quality lists $\mathcal{L}_{n+1}(v)$ for segment $n+1$ for multiple backward window lengths $v \in V$, where V denotes the set of window lengths used to calculate the spectrum. To include the drift to higher qualities we multiply the values of $\mathcal{L}_{n+1}(v)$ each with a corresponding quality weight

$$\alpha(q) = \left(\frac{r_1}{r_Q}\right)^{\frac{1}{Q-q+1}}. \tag{8}$$

Note that the bitrate ratio r_1/r_Q is less than one and that the weighting function $\alpha(q)$ is concave in $Q - q$.

We combine the weighted quality lists $\mathcal{L}^{\alpha}_{n+1}(v)$, i.e., the



{q^c_1, …, q^c_{n-1}, q^c_n} +MPD
get sustainable quality q^s_{n+1}
for q^s_{n+1} ∈ 1,...,Q
or
estimate fetch time
yes
no
unsustainable q^s_{n+1}
calculate spectrum H(n+1) over windows v∈V
apply weight α(q) for drift
calculate ordered quality list
choose q^s_{n+1}
maximizing the average of ℒ^α_{n+1}(v) over V

Figure 7: Spectrum based sustainable quality identification.

element wise product of the two vectors α and $\mathcal{L}_{n+1}(v)$, and calculate the average of $\mathcal{L}^{\alpha}_{n+1}(v)$ over multiple v. The basic adaptation algorithm sets the so-called *chosen* quality for the next segment q^C_{n+1} equal to the *sustainable* quality q^s_{n+1} which minimizes the average

$$\sum_{v \in V} \frac{1}{|V|} \mathcal{L}^{\alpha}_{n+1}(v), \tag{9}$$

where $|V|$ denotes the cardinality of V. In the following, we use $V = \{4, 8, 16\}$ segments if not stated otherwise.

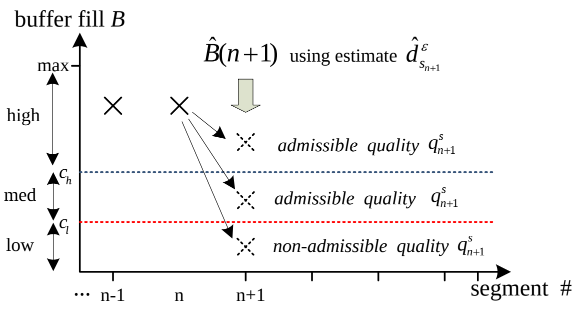
4.2.2 Buffer guidance - latent fallback

Now we turn our attention to relax the restrictive buffer constraint from above that we do not fetch qualities that lead to buffer drain. In general, our aim is to stream a movie in highest sustainable quality while minimizing the quality variations. Conceptually, a quality adaptation strategy that tries to keep the buffer filling fixed will eventually *follow* the variations of the available bandwidth. Hence, we use a buffer guidance approach to complement the spectrum based adaptation from Sect. 4.2.1. Here, we allow the playout buffer B to drain at most by a certain amount whenever the available bandwidth decreases and the current quality level becomes unsustainable. This latent fallback strategy allows the player to sacrifice buffer filling to maintain unsustainable but spectrum minimizing quality levels for short periods of time to be able overcome temporary available bandwidth fluctuations. This, however, is only possible when the buffer filling is high enough to minimize the risk of rebuffering. In the following we describe the details of this algorithm.

As depicted in Fig. 8, we divide the playout buffer into three areas, "low", "medium" and "high". Similar divisions have been introduced in [16] and `VLC` [21]. In our case, we only allow latent fallback when the buffer is in the "high" region. We mark the buffer division by cutoff percentages c_l and c_h where the subscripts stand for the lowest and highest area. When the buffer filling is above c_h we calculate the ordered quality lists $\mathcal{L}^{\alpha}_{n+1}(v)$ as in Sect. 4.2.1 but we *only* mark qualities as unsustainable, that lead to a buffer drain below c_l. In other words, we mark qualities q as unsustainable only when

$$\hat{B}(n+1) := B(n) - \hat{\delta}(n+1, q) < c_l, \tag{10}$$

where $\hat{\delta}(n+1, q) := \hat{t}_{n+1,q} - X$ is the estimated buffer drain due to fetching segment $n+1$ in quality level q. We keep



buffer fill B
B̂(n+1) using estimate d̂^ε_{s_{n+1}}
max
high
med
low
c_h
c_l
admissible quality q^s_{n+1}
admissible quality q^s_{n+1}
non-admissible quality q^s_{n+1}
... n-1
n
n+1
segment #

Figure 8: Estimated buffer drain $\hat{\delta}(n+1, q) = B(n) - \hat{B}(n+1)$ determines the admissibility of a proposed quality q^s_{n+1}. Latent fallback is viable only if the buffer filling is above c_h.

applying the latent fallback algorithm as long as the buffer filling is above c_h. As soon as the buffer filling is equal or below c_h we return to the buffer constraint from Sect. 4.2.1.

4.2.3 Player states

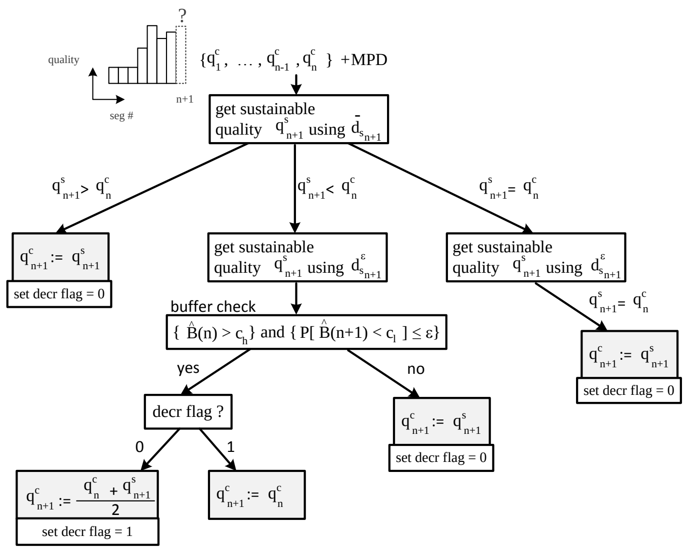
In steady state we differentiate between three states in which the player can be, which we denote as **(i) decreasing** , **(ii) steady**, **(iii) increasing**. These states simply describe the relation between the current *chosen* segment quality q^C_n and the *sustainable* quality level that is calculated for the next segment to be fetched q^s_{n+1}. Based on its state the player decides on the *chosen* segment quality for the next segment, i.e., q^C_{n+1}. In the following, we describe the steps that are common to all states before delving into the particular details of every state. Fig. 9 shows an overview of the entire algorithm.

Before fetching a new segment we always calculate the corresponding *sustainable* quality level q^s_{n+1} as described in Sect. 4.2.1. In particular, q^s_{n+1} is calculated as the quality which minimizes the spectrum, technically (9), while meeting the stringent buffer constraint $\hat{t}_{n+1} \leq X$ from Sect. 4.2.1. To this end, we first calculate the estimated fetch time for segment $n+1$ in quality q for $q \in \{1, \ldots, Q\}$ as

$$\hat{t}_{n+1,q} = \frac{s_{n+1,q}}{\bar{d}_{s_{n+1,q}}} \tag{11}$$

where $\bar{d}_{s_{n+1,q}}$ denotes the average download rate estimated empirically from Fig. 6b (similar to Fig. 4) given the size of the next segment $s_{n+1,q}$. Hence, as stated above we find q^s_{n+1} that minimizes (9) and compare it to q^C_n to detect the current player state. In the following, we describe the steps carried out when the different player states are detected:

Decreasing: In the decreasing state we detect that the sustainable quality level is less than the current chosen quality, $q^s_{n+1} < q^C_n$. Here we first invoke the latent fallback technique that is described in Sect. 4.2.2 to avoid rapid quality changes in exchange for buffer filling. To this end, we recalculate the sustainable quality q^s_{n+1} that minimizes (9), however, using the ε percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$ from (7). In the following, we set the percentile $\varepsilon = 0.2$ to strike a balance between smoothness and responsiveness. Next, we consider the buffer filling as in Sect. 4.2.2. If the buffer filling is higher than c_h, i.e., we have enough segments buffered, we may sacrifice buffer filling for holding the quality level q_n. Hence, as long as the estimated buffer underflow probability after fetching the next segment is less than ε, i.e.,



quality
seg #
n+1
?
$\{q^c_1, \ldots, q^c_{n-1}, q^c_n\}$ +MPD
get sustainable quality q^s_{n+1} using $\bar{d}_{s_{n+1}}$
$q^s_{n+1} > q^c_n$
$q^s_{n+1} < q^c_n$
$q^s_{n+1} = q^c_n$
$q^c_{n+1} := q^s_{n+1}$
set decr flag = 0
get sustainable quality q^s_{n+1} using $d^\varepsilon_{s_{n+1}}$
buffer check
$\{\hat{B}(n) > c_h\}$ and $\{P[\hat{B}(n+1) < c_l] \le \varepsilon\}$
yes
no
decr flag ?
0
1
$q^c_{n+1} := \frac{q^c_n + q^s_{n+1}}{2}$
set decr flag = 1
$q^c_{n+1} := q^c_n$

Figure 9: High-level sketch of the SQUAD algorithm. The calculation block to get the sustainable quality is given in Fig. 7. The buffer drain consideration in the decreasing player state is given in Fig. 8.

$$P\left[\hat{B}(n+1) < c_l\right] \le \varepsilon, \tag{12}$$

we set the *chosen* quality for the next segment as the average $q^C_{n+1} := (q^C_n + q^s_{n+1})/2$ with q^s_{n+1} calculated using the percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$. Note that we do not perform this averaging procedure more than once during one decreasing period, since the idea here is to hold a moderate quality level as long as the buffer filling permits. As described in Sect. 4.2.2, as soon as the current buffer filling q^C_n falls below c_h we set the chose quality level as $q^C_{n+1} := q^s_{n+1}$. This procedure is depicted in Fig. 9.

Increasing: In the increasing state we observe that $q^s_{n+1} > q^C_n$ through the use of the average download rate $\bar{d}_{s_{n+1,q}}$. The increasing state denotes that the player has room for increasing the segment quality. In this case we set the chosen quality as $q^C_{n+1} := q^s_{n+1}$.

Steady: In the steady case we detect no change in the calculated sustainable quality level with respect to the previously fetched segment, i.e., $q^s_{n+1} = q^C_n$. In this case, we decide to be cautious and recalculate q^s_{n+1} using the percentile $\hat{d}^{\varepsilon}_{s_{n+1,q}}$. If q^s_{n+1} stays unchanged then we set the quality as $q^C_{n+1} := q^s_{n+1}$ if we detect $q^s_{n+1} < q^C_n$ then we undergo the same procedure as in the decreasing state.

5. EXPERIMENTAL EVALUATION

In the following, we conduct a number of experiments in a controlled testbed, the GENI testbed, as well as in the public Internet to evaluate the performance of SQUAD.

For all experiments we make use of an excerpt of the `BigBuckBunny` dataset [18] that comprises a video that is 300 seconds long and an MPD that describes attributes of the video. We extended the MPD file by providing the size of each segment in each of the available quality levels. The quality bitrates available in this MPD are the following {`0.09, 0.13, 0.18, 0.22, 0.26, 0.33, 0.59, 0.79, 1.03, 1.24, 1.54, 2.48, 3.52, 4.21`} Mbps.

For better judgement of the performance of SQUAD we compare its performance with the ones of three additional algorithms, which we are denoted "VLC", "SARA" and "Buffer-based". We briefly describe them in the following:

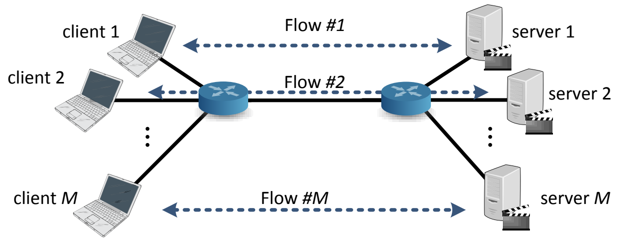


Figure 10: Butterfly evaluation topology.

VLC: The first algorithm we decided to chose for comparison is a basic quality adaptation algorithm from [21]. The algorithm works as follows: The client takes a pair of information into account when deciding on the quality of the next segment to be fetched, i.e., (*i*) the current playout buffer filling, and (*ii*) the empirical download rate of the previous segment. If the current buffer filling is < 25% of the maximum buffer size then the client always fetches the lowest quality. In case the buffer filling is > 25% the client greedily downloads the next segment at the highest sustainable quality. In case the buffer is full the client idles.

SARA: The second algorithm we use for comparison has been proposed in [16] and coined segment aware rate adaptation (SARA). The algorithm predicts the time required for fetching a segment based on its size and the available bandwidth estimate through a weighted harmonic mean. Further, SARA selects the bitrate depending on the current buffer filling and drops to the lowest bitrate if the buffer filling falls below a certain threshold. Note that the SARA implementation accompanying [16] uses non-persistent HTTP by default.

SQUAD: Our proposed spectrum-based quality adaptation algorithm is explained in detail in Sect. 4.

Buffer-based: This algorithm is denoted BBA-0 in [14] and is implemented as part of the Python DASH client emulator accompanying [16]. In a nutshell, the algorithm defines a class of functions that map current buffer occupancy to a quality bitrate (denoted rate map) to avoid unnecessary rebuffering and maximize the average video rate. This algorithm was part of a wide-scale Netflix experiment in [14].

5.1 GENI

The GENI (Global Environment for Networking Innovation) testbed is a distributed virtual laboratory sponsored by the U.S. National Science Foundation (NSF). It allows researchers to obtain a virtualized and isolated slice of compute, storage, and networking resources for the development and validation of new approaches in networking and distributed systems [8, 9]. GENI allows the setup of larger and potentially wide-area topologies.

For the evaluation in the GENI testbed we create a slice that comprises a butterfly topology as shown in Fig. 10. In the following experiments, we stream the DASH video from server i to client i for $i \in \{1, 2, 3\}$. For the experiments with one DASH flow we utilize server 1 and client 1 and for the cross traffic flow we utilize server 2 and client 2. All links possess a capacity of 10 Mbps.

We divide the GENI experiments into two categories: (*i*) Single DASH algorithms vs. UDP cross traffic, (*ii*) concurrent DASH clients with no cross traffic. In the first set of

Algorithm	average quality bitrate [Mbps]				# of quality switches				spectrum H			
	VLC	SARA	SQUAD	BBA	VLC	SARA	SQUAD	BBA	VLC	SARA	SQUAD	BBA
UDP-U	3.67	3.51	3.24	3.20	20	36	11	60	2277	3590	1031	6214
UDP-W	3.78	3.84	3.86	3.54	10	30	10	55	1216	3001	1053	6067
UDP-ONOFF	3.74	3.96	4.02	3.33	38	26	4	78	4691	2259	202	7652

Table 1: QoE metrics for the UDP experiments in Fig. 11 to 14. Higher average quality bitrate is better. Lower # of quality switches and lower spectrum values are better.

experiments, i.e., in Fig. 11 to 14 we run (v) contiguous UDP cross traffic of 8 Mbps for 2 minutes (available bandwidth has U-shape), (vv) two-level UDP cross traffic of 8 and 5 Mbps (available bandwidth has W-shape) and (vvv) alternating ON-OFF UDP cross traffic of 8 Mbps in the ON state. Fig. 11 to 14 depict 5 minute sample runs for each of the studied algorithms showing the following: 1) The empirical segment download rate (denoted in figures as seg. DL rate), 2) the quality bitrate (important QoE metric), 3) the buffer filling over time, as well as, 4) instantaneous rate measurements of the cross traffic (shown as crosses) and of the DASH flow (shown as dots). In all following figures, the y-axis denotes the rate in Mbps *and* for the buffer filling curves it denotes the buffer length in segments. Note that one segment is 2 seconds long in this dataset and that we set the maximum buffer size to 30 seconds, i.e., 15 segments.

From Fig. 11 to 14 we deduce the following observations: First, the VLC algorithm is highly aggressive in choosing the quality bitrate which may substantially drain its playout buffer, enforcing it to significantly reduce the quality bitrate when the buffer reaches 25%. This leads to high quality jump magnitudes which are detrimental to QoE. Secondly, SARA introduces many oscillations of the fetched quality bitrate around the available bandwidth which is harmful to the QoE. On the contrary SQUAD performance is smooth: With the latent fallback and the percentile, respectively, average rate estimation it holds the quality bitrate over temporary available bandwidth fluctuation. The buffer-based algorithm in Fig. 14 possesses many unnecessary quality switches. The quantitative results of the average quality bitrate, the number of quality jumps, as well as, the spectrum H are given in Tab. 1. These metrics show that SQUAD provides a significant QoE improvement as seen by the strong reduction in the number of quality jumps [31] while sacrificing little or no average quality bitrate. SQUAD also outperforms its competitors in minimizing the variation of the quality bitrates (spectrum).

The second set of experiments shown in Fig. 15 to 17 depict the measurement results for 3 concurrent homogeneous clients without cross traffic. Fig. 18 shows measurement results for 2 concurrent clients running the VLC and SQUAD algorithms. Fig. 15 to 17 pictorially show that clients running VLC or SQUAD fairly share the bottleneck bandwidth. Concurrent SARA clients suffer from fairness issues. This phenomenon can be directly explained knowing that SARA uses non-persistent HTTP by default. We ran modified SARA measurements with persistent HTTP connections, where the client suffered only from quality oscillations as shown in Fig. 11b. The comparison in Fig. 18 shows that VLC is more aggressive than SQUAD which uses smooth rate estimates from Sect. 4.2. Observe that VLC

Algorithm	jump magnitude [Mbps]		# of switches		quality bitrate [Mbps]	
	avg.	CI	avg.	CI	avg.	CI
VLC	1.45	± 0.05	34.9	± 3.1	3.1	±0.02
SARA	0.5	± 0.04	67.6	± 4.6	0.8	± 0.03
SQUAD	0.58	± 0.05	4.6	± 0.5	2.3	± 0.1

Table 2: QoE metrics for the streaming experiment US-Germany. The table also includes 0.95-confidence intervals.

consistently overestimates the available bandwidth leading to strong variations of the quality bitrate and quick buffer depletion.

5.2 Internet

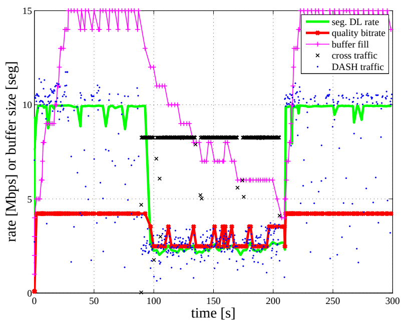
Besides evaluating SQUAD performance in controlled testbeds, we are also interested in its performance in the "wild". We perform a series of experiments over the public Internet, where we use a web server at a University in Germany which hosts the DASH movie dataset described in the beginning of the section. The client is located in a residential home in the east coast region of the US and runs the SQUAD algorithm amongst the other DASH algorithms we described earlier in this section. Fig. 19 shows smooth playback by SQUAD which selects the right quality bitrate. Note that in this case a higher quality bitrate, that is chosen by VLC, is unsustainable and leads to buffer depletion. VLC overestimates the available bandwidth and suffers from severe quality switches (see the average jump magnitude and the number of quality switches in Tab. 1). The SARA algorithm struggles due to non-persistence.[1] Figure 19 clearly demonstrates how SQUAD outperforms VLC and SARA in a real-world environment.

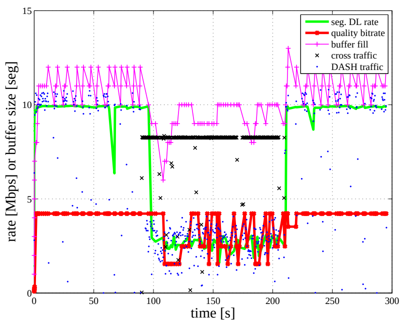
6. RELATED WORK

Due to its popularity, DASH has been extensively studied in recent publications. The related work for DASH can be coarsly divided into two categories, i.e., (i) large-scale measurement studies showing real-world streaming behavior of DASH, and (ii) studies that propose new quality adaptation mechanisms to improve the DASH streaming performance. In the following, we review related work that is most relevant for this paper and highlight the differences to SQUAD.

Adaptive Video Streaming Measurements: Large-scale studies of DASH performance have been conducted in commercial video streaming platforms such as Hulu,

[1]We include further measurement results, e.g., with modified SARA, into the technical report submitted as auxiliary material.


(a) VLC.


(b) SARA.

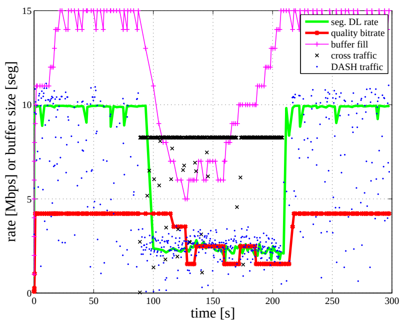
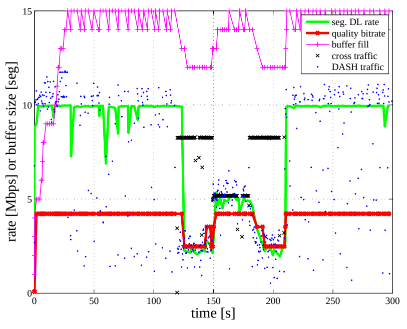
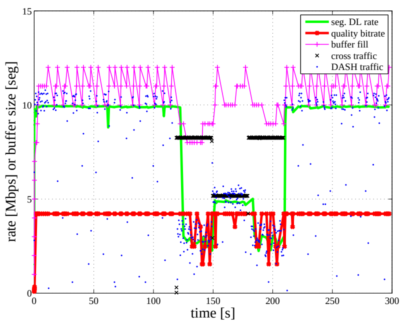

(c) SQUAD.

Figure 11: Quality bitrate with UDP-U cross traffic.


(a) VLC.


(b) SARA.

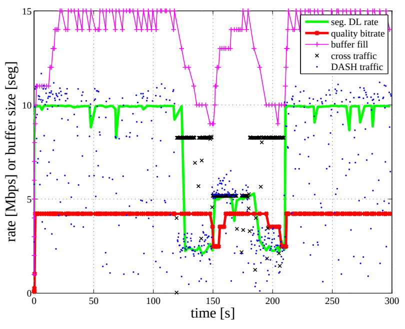
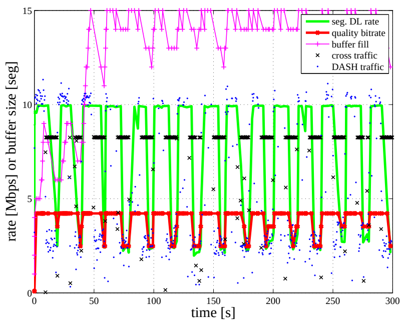
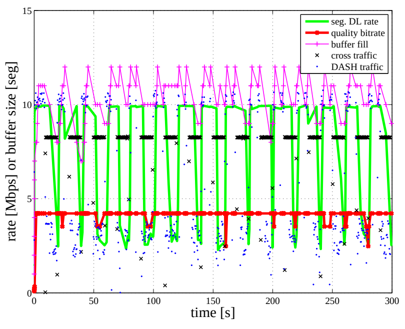

(c) SQUAD.

Figure 12: Quality bitrate with UDP-W cross traffic.


(a) VLC.


(b) SARA.

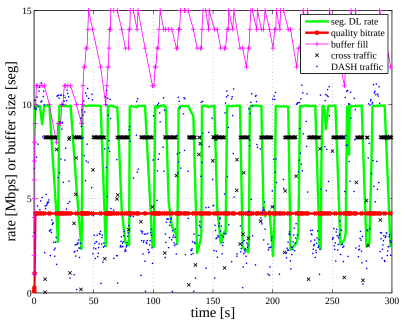
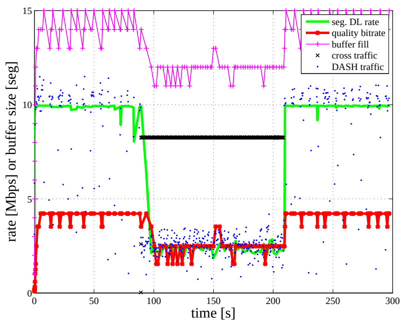
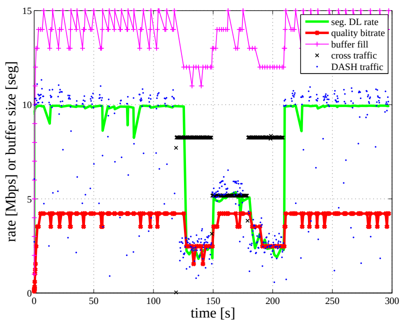

(c) SQUAD.

Figure 13: Quality bitrate with UDP ON-OFF cross traffic.


(a) UDP-U.


(b) UDP-W.

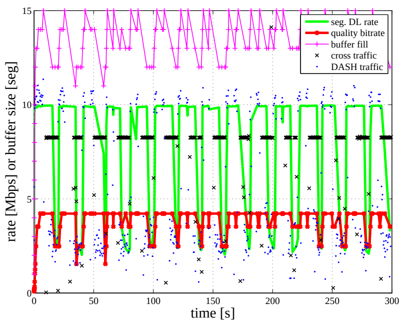
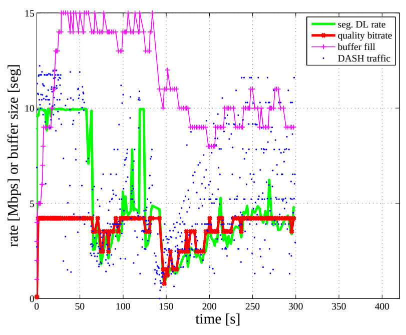
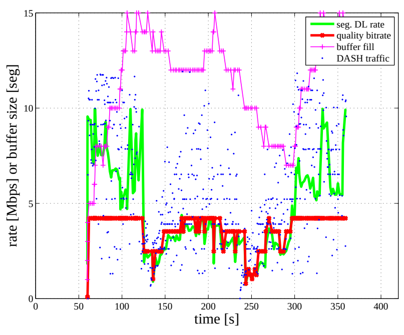

(c) UDP ON-OFF.

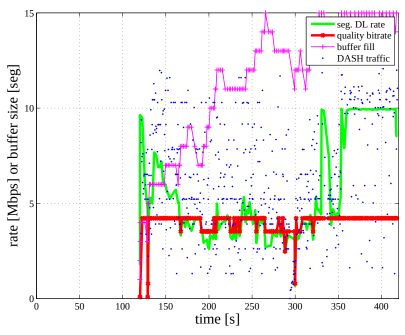
Figure 14: Quality bitrate with Buffer-based [14] algorithm. (Implementation accompanying [16].)



(a) Client1

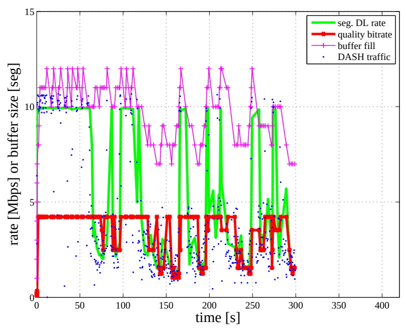


(b) Client2

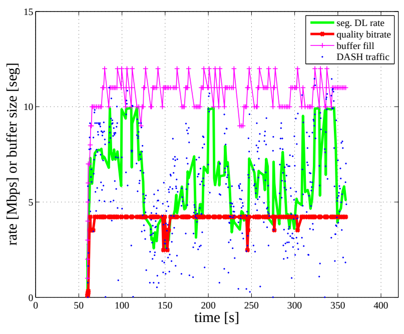


(c) Client3

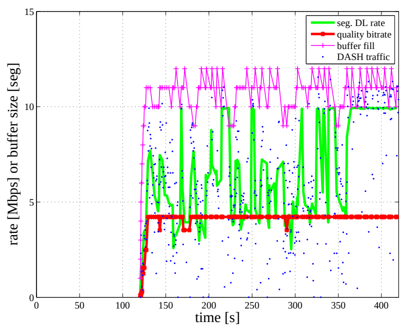
Figure 15: Concurrent VLC clients. Clients start with 60 sec relative delay.



(a) Client1

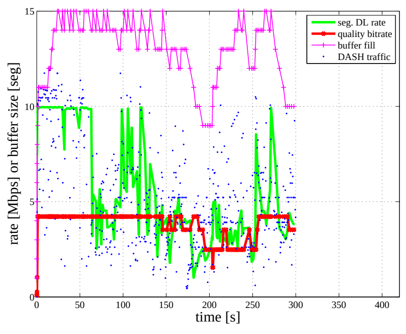


(b) Client2

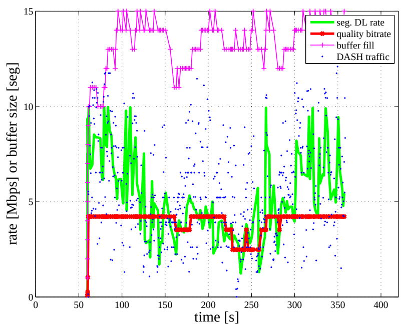


(c) Client3

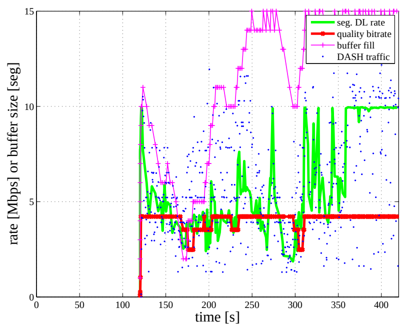
Figure 16: Concurrent SARA clients. Clients start with 60 sec relative delay.



(a) Client1

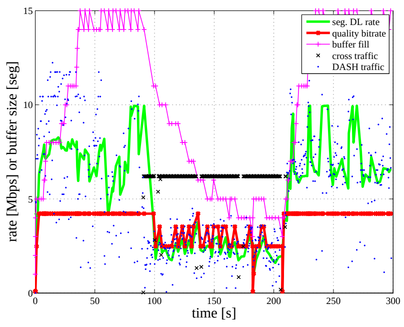


(b) Client2

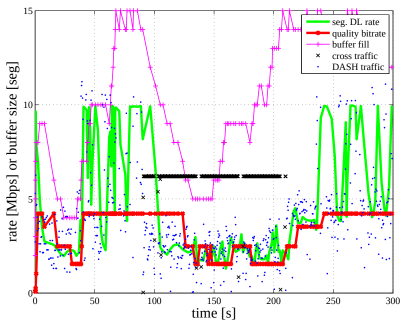


(c) Client3

Figure 17: Concurrent SQUAD clients. Clients start with 60 sec relative delay.

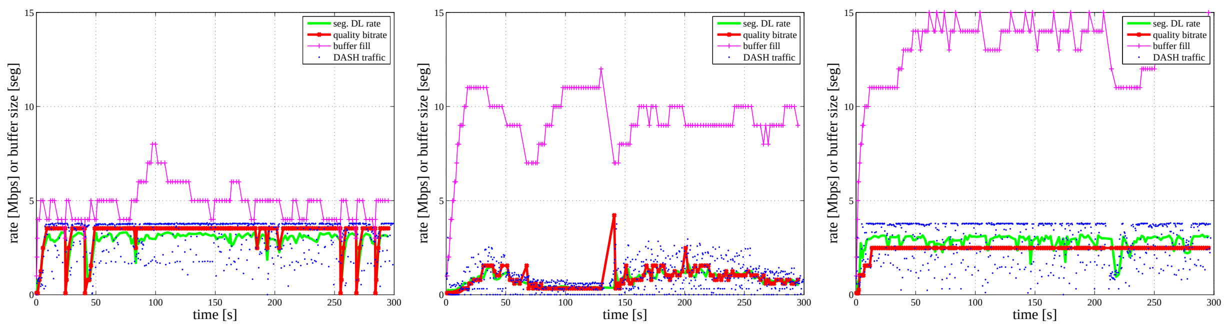


(a) VLC



(b) SQUAD

Figure 18: Concurrent clients, VLC vs. SQUAD. When the cross traffic is on, VLC overestimates the available bandwidth leading to oscillations, buffer depletion and high quality jump magnitudes. SQUAD playback is smooth with little variations.



(a) VLC (b) SARA (c) SQUAD

Figure 19: Quality bitrate for streaming scenario: US-Germany.

Netflix and Vudu [13, 7, 14]. Similar measurement studies also covered aspects of the transmission behavior of video clients [6], DASH network traffic characteristics [23], and DASH QoE [11, 22]. These large-scale measurement studies point out some of the real-world difficulties that face ABR streaming such as the high variability of the end-to-end throughput, the inaccuracy of client based rate estimation, as well as, rebuffering risk due to non-preemptive segment download. These factors make a purely rate-base quality adaptation approach highly volatile.

Rate Selection Policies: As the DASH standard does not specify how to perform rate adaptation, many research groups studied the impact of different rate adaptation algorithms on DASH performance. The existing body of work on DASH quality adaptation has two sources of information, i.e., the buffer fill and the available bandwidth estimate. Proposed algorithms usually use one of these two sources as a main information source and the other one to cover corner cases. The work in [14] uses measurements from a commercial video streaming platform to assert that it is sufficient to mainly use the client buffer fill level to determine the quality of the next segment to download. Here, the authors find that it is important to obtain available bandwidth estimates only during the startup phase.

The authors of [26] propose a traffic shaping mechanism to alter the packet interarrival times, hence, improve the accuracy of available bandwidth estimates by reducing traffic burstiness. The work in [25] proposed a PID controller for rate adaptation which takes the buffer filling into account to refine the available bandwidth estimates. Similarly, the authors of [10] propose a control theoretic approach to stabilize the buffer filling at certain level.

Studies that investigate the impact of different segments sizes and lengths include [5, 16]. In Sect. 5 we provided a brief sketch of the SARA algorithm from [16], which takes into account that segment sizes (in bytes) may differ widely even within the same quality level.

Closest to our work are two existing approaches that take QoE into account when performing DASH rate adaption. The first approach, that is denoted PANDA (for probe-and-adapt) [19, 20], presents a buffer filling based adaptation algorithm that solves the quality selection optimization problem with respect to an α-fairness objective using a dynamic programming approach. The authors use peak signal-to-noise ratio (PSNR) to capture QoE. Our work differs from this approach as we do not only aim to maximize the average quality but also minimize the number of quality changes. We use the variation of the quality bitrates, denoted as spectrum, as a metric to capture QoE, since it is known, for example from [31] that it is superior to PSNR.

The second approach that is closely related to ours is given in [30], where the authors propose a QoE metric that is a weighted combination of the average video quality, the average quality variation, the rebuffering time and the startup delay. The authors formulate the rate selection problem as a stochastic optimal control problem. Assuming stability of network conditions on *known* finite time horizons, the proposed algorithm uses a model predictive control approach to optimize the QoE metric mentioned above. The algorithm in [30] is based on an offline section, which does offline optimization (using CPLEX) for different scenarios, and an online section, which comprises table lookups of precalculated solutions. A similar approach has been leveraged in [29] which formulates the rate selection problem as a Markov decision process and uses dynamic programming to find an optimal solution. For these methods to find the optimal rate selection policy, strong assumptions have to be made on the statistics and predictability of the network conditions.

7. CONCLUSIONS

Motivated by critical observations of operation characteristics of dynamic adaptive streaming over HTTP (DASH) we provide in this paper a QoE tailored quality adaptation algorithm denoted SQUAD. SQUAD takes multiple QoE metrics into account, i.e., the average quality bitrate and most importantly its variation. A quick dissection of the DASH control loop over TCP shows the discrepancy of the available bandwidth estimation time scale between the quality adaptation and the underlying transport protocol. Bearing this in mind we construct SQUAD such that it uses rate estimates on the appropriate time scales. We test our Player implementation against multiple quality adaptation algorithms of different objectives in a controlled network environment, as well as, in a cross-Atlantic streaming experiment. The experiments show that by sacrificing little or no average quality bitrate, SQUAD provides significantly better QoE in terms of quality switching frequency and magnitude.

8. REFERENCES

[1] Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html. Accessed: 2015-12-09.

[2] Apple HTTP Live Streaming. https://developer.apple.com/resources/http-streaming/. Accessed: 2015-11-09.

[3] Global Internet phenomena report 2h 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf. Accessed: March, 30, 2015.

[4] Microsoft Smooth Streaming. http://www.iis.net/downloads/microsoft/smooth-streaming. Accessed: 2015-11-09.

[5] Optimal segment length for adaptive streaming formats like MPEG-DASH & HLS. http://www.dash-player.com/blog/2015/04/using-the-optimal-segment-length-for-adaptive-streaming-formats-like-mpeg-dash-hls/. Accessed: 2015-10-14.

[6] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *Proceedings of NOSSDAV*, pages 9–14, 2012.

[7] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of MMSys*, pages 157–168, 2011.

[8] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds – Part I.

[9] M. Berman, P. Demeester, J. W. Lee, K. Nagaraja, M. Zink, D. Colle, D. K. Krishnappa, D. Raychaudhuri, H. Schulzrinne, I. Seskar, and S. Sharma. Future Internets escape the simulator. *Commun. ACM*, 58(6):78–89, May 2015.

[10] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of MMSys*, pages 145–156, 2011.

[11] F. Fund, C. Wang, Y. Liu, T. Korakis, M. Zink, and S. Panwar. Performance of DASH and WebRTC video services for mobile users. In *IEEE Packet Video Workshop (PV)*, pages 1–8, Dec 2013.

[12] G. Giambene. *Queuing Theory and Telecommunications: Networks and Applications.* Springer US, 2005.

[13] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proceedings of IMC*, pages 225–238, 2012.

[14] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of ACM SIGCOMM*, pages 187–198, 2014.

[15] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Transactions on Networking*, 11(4):537–549, Aug. 2003.

[16] P. Juluri, V. Tamarapalli, and D. Medhi. Sara: Segment-aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *IEEE ICC QoE-FI Workshop*, June 2015.

[17] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *Proceedings of IMC*, pages 211–224, 2012.

[18] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of MMSys*, pages 89–94, 2012.

[19] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming video over HTTP with consistent quality. In *Proceedings of MMSys*, pages 248–258, 2014.

[20] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, April 2014.

[21] C. Müller and C. Timmerer. A VLC media player plugin enabling dynamic adaptive streaming over HTTP. In *Proceedings of the ACM Conference on Multimedia*, pages 723–726, 2011.

[22] O. Oyman and S. Singh. Quality of experience for HTTP adaptive streaming services. *IEEE Communications Magazine*, 50(4):20–27, April 2012.

[23] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proceedings of CoNEXT*, pages 25:1–25:12, 2011.

[24] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the Internet. *IEEE MultiMedia*, 18(4):62–67, April 2011.

[25] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of CoNEXT*, pages 109–120, 2012.

[26] B. Villa and P. Heegaard. Group based traffic shaping for adaptive HTTP video streaming by segment duration control. In *Proceedings of AINA*, pages 830–837, March 2013.

[27] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of OSDI*, pages 255–270, Dec. 2002.

[28] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *SIGCOMM Computer Communication Review*, 41(1):38–44, Jan. 2011.

[29] S. Xiang, L. Cai, and J. Pan. Adaptive scalable video streaming in wireless networks. In *Proceedings of MMSys*, pages 167–172, 2012.

[30] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of ACM SIGCOMM*, pages 325–338, 2015.

[31] M. Zink, J. Schmitt, and R. Steinmetz. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia*, 7(1):75–84, Feb 2005.