OSCILLATION COMPENSATING DYNAMIC ADAPTIVE STREAMING OVER HTTP

Christopher Mueller[†]*, Stefan Lederer*[†]*, Reinhard Grandl*[†]*, and Christian Timmerer*[†‡*]

[†]bitmovin GmbH (http://www.bitmovin.net)
[‡]Alpen-Adria-Universität Klagenfurt, Institute of Information Technology (ITEC)
Klagenfurt am Wörthersee, Austria
[†]{*firstname.lastname*}@bitmovin.net, [‡]christian.timmerer@itec.aau.at

ABSTRACT

Streaming multimedia over the Internet is omnipresent but still in its infancy, specifically when it comes to the adaptation based on bandwidth/throughput measurements, clients competing for limited/shared bandwidth, and the presence of a caching infrastructure. In this paper we present a buffer-based adaptation logic in combination with a toolset of client metrics to compensate for erroneous adaptation decisions. These erroneous adaptation decisions are due to insufficient network information available at the client and issues introduced when multiple clients compete for limited/shared bandwidth and/or when caches are deployed. Our metrics enable the detection of oscillations on the client – in contrast to server-based approaches – and provide an effective compensation mechanism. We evaluate the proposed adaptation logic, which incorporates the oscillation detection and compensation method, and compare it against a throughput-based adaptation logic for scenarios comprising competing clients with and without caching enabled. In anticipation of the results, we show how the presented metrics detect oscillation periods and how such undesirable situations can be compensated while increasing the effective media throughput of the clients.

Index Terms— MPEG-DASH, Dynamic Adaptive Streaming over HTTP, Oscillation Detection and Compensation, Clients Competing for Bandwidth

1. INTRODUCTION

The transport of multimedia content over the Internet has gained momentum and entered our daily lives, whether it be a major live event or on-demand video. For example, we see people accessing live sport events or watching their favorite TV series on a plethora of devices ranging from high-resolution, well-connected TV sets to smart mobile devices with limited display and network capabilities. All of these use cases have something in common where the content is delivered over the Internet and on top of the existing infrastructure.

The basic concept of todays' HTTP-based multimedia streaming solutions is to provide multiple versions of the same content (e.g., different bitrates), chop these versions into (small) segments (e.g., two seconds), and let the client decide which segment (of which version) to download next, based on its context (e.g., available bandwidth). Typically, the relationship between the different versions is described by a manifest, which is provided to the client prior to the streaming session. The ISO/IEC MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard specifies representation formats for both the manifest and segments [1]. For the manifest, DASH defines the XML-based Media Presentation Description (MPD) representing the data model, which is aligned with existing, proprietary solutions.

The most challenging part of a DASH client implementation is the component that determines which segment to download next. This component is often referred to as *adaptation algorithm/logic*. After receipt of the MPD, it basically analyzes the available representations (e.g., bitrates, resolutions) given the current context (e.g., bandwidth, display size) and starts downloading the segments accordingly. In case the context changes (e.g., due to a drop of the available bandwidth), the client may switch to another representation that is suitable for the new context. The actual switching is typically done at segment boundaries and, in general, the behaviour of the adaptation logic has a direct influence on the system performance [2–4]. The system performance depends on a number of metrics which can be both of objective and subjective nature. Our focus is on objective metrics, which may be fine-tuned through subjective quality assessments. In a typical deployment, multiple clients may compete with each other and may introduce unwished issues, specifically when proxies/caches are deployed, which is often the case in combination with CDNs [5]. In such a deployment oscillations may occur where clients often switch between different versions representing different qualities resulting in poor Quality of Experience (QoE) and reduced system performance. Proxy-based approaches may perform traffic shaping towards multiple clients [6] while others prefer server-based traffic shaping to stabilize oscillating clients [7]. Additionally, on-demand rate adaptation [8] and request re-writing [9] are other proxy-based solutions but do not take into account the issue when clients compete for bandwidth.

In this paper, we present a novel, client-based adaptation logic based on [5] that enables oscillation detection and compensation in scenarios where multiple clients compete for (limited) bandwidth. We show that in these scenarios oscillations occur and, if not considered by the adaptation logic, gets even worse when caches are involved. Therefore, we present a set of *metrics and tools* which are used by our proposed adaptation algorithm that compensates these oscillations while adapting to dynamic context changes and maximizing link utilization (i.e., media throughput at the client) as well as maintaining an overall smooth viewing experience (i.e., no stalls). In anticipation of the results, our proposed adaptation algorithm successfully detects oscillation – as opposed to pure quality switching – and smooths the detected oscillations, e.g., by delivering a lower bitrate for half the time and higher bitrate for the other half of the time. Finally, our approach is purely client-centric and works over the top without modifying the underlying infrastructure or putting

*The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no 610370, ICoSOLE ("Immersive Coverage of Spatially Outspread Live Events, http://www.icosole.eu).

any requirements for content and service providers.

The remainder of the paper is organized as follows. Section 2 provides the problem statement showing that oscillations occur under given conditions. Section 3 defines metrics and tools which are used for our adaptation logic described in Section 4. Experimental results are covered in Section 5 and Section 6 comprises conclusions as well as points out future work.

2. PROBLEM STATEMENT

Dynamic adaptive streaming over HTTP allows for a flexible and scalable deployment of media ecosystems [1] as the client encapsulates the entire streaming logic and no centralized controller is needed, also thanks to the stateless design of HTTP. Additionally, this kind of streaming approach enables the reuse of the already deployed Internet infrastructure comprising proxies, caches, and content distribution networks (CDNs). However, the actual nature of this ecosystem, which enables switching between individual quality levels without a centralized controller, may also introduce drawbacks. Therefore, in this section we highlight the problems that might occur when multiple clients compete for bandwidth in a DASH-like streaming ecosystem including an experimental validation thereof.

The general assumption that TCP will accommodate the case when multiple clients compete for bandwidth is invalidated in [5, 7] where clients begin to oscillate, specifically by continuously switching between different quality levels. In particular, Akshabi et al. [7] propose a server-based solution to mitigate the oscillation effect but this may eliminate some major benefits such as the stateless design and the usage of ordinary HTTP servers. Furthermore, in a large-scale deployment where clients may request segments from multiple sources (servers, proxies, CDN nodes), the adoption of a server-based solution increases deployment costs and decreases scalability as segment sources need to exchange additional information about the oscillation state. In comparison, Mueller et al. [5] identify client oscillations in conjunction with proxy caches and propose a client-centric approach that does not require any modifications on the existing infrastructure.

In order to validate the problem statement we have performed experiments to demonstrate the oscillation with and without caches present. Our experimental setup consists of a content server that provides an excerpt of 300s from the Big Buck Bunny sequence in six different media bitrates with constant bitrate (350, 700, 1300, 1900, 2500, and 3400 kbps) and a segment length of 2 seconds. Two clients are connected to the content server through a cache and a bottleneck that simulates a predefined but varying available bandwidth (2800-3200 kbps) as shown in Figure 1. The results shown in this section are based on a throughput-based adaptation logic such as the one from Liu et al. [3] which takes the throughput measurements of the downloaded segment as a basis for the quality switching. That is, if the throughput is higher than the media bitrate (according to the manifest/MPD), a switch to the next higher representation is performed, and if the throughput is lower than the media bitrate, a switch to the next lower representation is done. This decision can be done on a per segment basis. The same setup has been used for the actual experimental results of the proposed adaptation logic (see Section 5).

First, we present the results of the throughput-based adaptation algorithm with and without cache. Figure 2 shows the results for client 1 (in red on the left) and client 2 (in blue on the right) without a cache. The upper part shows the requested media bitrate and the lower part depicts the buffer fill state, both over the entire length of the session (300s). Client 1 starts 10 seconds before client 2 and,

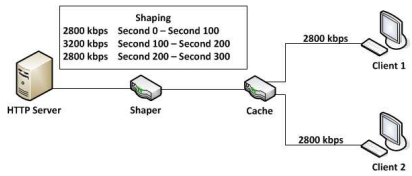


Fig. 1: Experimental Setup.

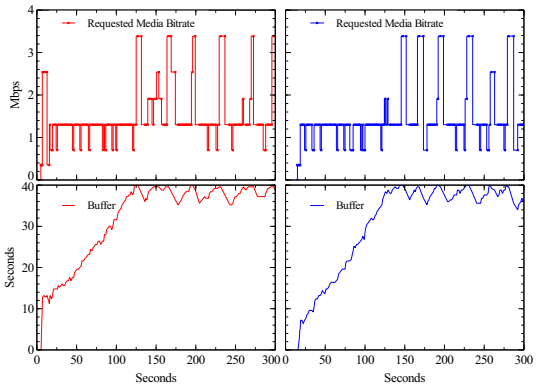


Fig. 2: Throughput-based Adaptation without Cache for Client 1 (red/left) and Client 2 (blue/right).

thus, is able to fully utilize the bottleneck in the beginning, i.e., it selects the quality level with 2500 kbps. As soon as client 2 starts its session, both clients have to share the bottleneck bandwidth and, if done in a fair manner, each client will get a share of about 1400 kbps of the available bandwidth. The representation with 1300 kbps comes very close but throughput measurements are sometimes above and sometimes below this bitrate due the behavior of the underlying TCP implementation. Thus, the throughput measurements in both clients lead to an oscillation between the 700 and 1300 kbps respectively. This oscillation decreases between second 100 and 200 after increasing the bottleneck bandwidth to 3200 kbps.

In this scenario, no quality level is available that fits exactly the available bandwidth. Therefore, both clients select a lower quality level which fills up their buffers until they are full which result in the ON/OFF state already shown by Akshabi et al. [7]. Once the buffer is full, the throughput measurements return an erroneous value as the client implementation (on the application layer) cannot continuously read data from the transport layer while TCP keeps receiving data that will be buffered within the kernel. Additionally, with this behavior both clients influence each other in a negative way. For example, in second 125 client 1 stops the download process (as the buffer is full) which leads to an oscillation on client 2. After client 1 continuous with the download, client 2 has to switch to a lower quality level. The same behavior occurs in second 160 but here client 2 stops its download process and client 1 starts to oscillate.

Figure 3 shows the behavior of the throughput-based adaptation logic with a cache which should enable a more efficient usage of the bottleneck. However, when comparing the results with the experiment without a cache, the adaptation behavior on both clients gets worse and both clients frequently oscillate between 1300 and 2500 kbps. This problem is also shown in [5] with different throughput-based adaptation logics (i.e., Microsoft Smooth Streaming and the DASH VLC plugin).

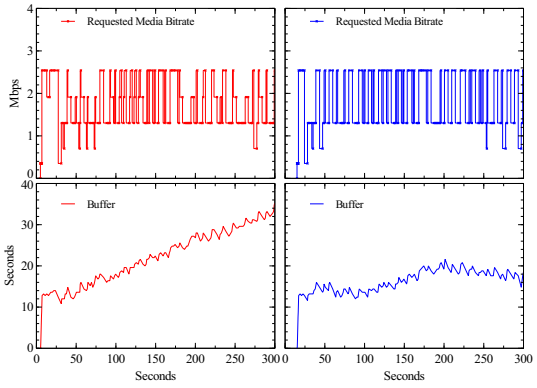


Fig. 3: Throughput-based Adaptation with Cache for Client 1 (red/left) and Client 2 (blue/right).

3. METRICS AND TOOLS

This section describes metrics to be used as an input to our adaptation logic in order to detect issues as highlighted in Section 2. Each metric can be seen as an individual tool that characterizes the adaptation or download process in a specific manner. We show how to combine these tools to efficiently detect and characterize oscillation and quality switching behavior in a way that allows for a smooth balancing of the streaming session. In the following, we define the metrics clustered depending on whether they are specific to the adaptation or the buffer state.

The metrics and tools are defined for a given time window comprising a set of segments s with bitrate θ of length t seconds denoted by $\Delta = \{(\theta_i, t_i), (\theta_{i+1}, t_{i+1}), ..., (\theta_{j-1}, t_{j-1}), (\theta_j, t_j)\}$ with $i, j \in \mathbb{N}$, and $i < j$. Hence, our metrics may be applied on a sliding window over recently retrieved segments and support variable segment length which is also supported by DASH.

3.1. Adaptation-specific Metrics and Tools

Equation 1 defines the *mean* μ of the recently retrieved segments as the average calculated from the media bitrate of segments θ_k and the length of each segment t_k. The size of the window may be adjusted with the parameters i and j, which are an index into Δ with i referring to the first segment and j to the last segment.

$$\mu_{i,j} = \frac{\sum_{k=i}^{j} \theta_k \times t_k}{\sum_{k=i}^{j} t_k} \tag{1}$$

Based on Equation 1 we define the *quality switching variance* σ as a modification of the variance. Therefore, we define Equation 2 which determines whether a segment should be considered for calculating the quality switching variance by returning 1 when bitrates of successive segments are different (i.e., $\theta_k \neq \theta_{k-1}$) which indicates a quality switching event or when there is only one segment retrieved (i.e., $|\Delta| = 1$).

$$\Gamma_k = \begin{cases} 1 & \text{if } |\Delta| = 1 \\ 1 & \text{if } |\Delta| > 1, \theta_k \neq \theta_{k-1} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Equation 3 defines the quality switching variance σ over a given time window starting from segment i to segment j taking only quality switching events into account thanks to Γ from Equation 2. Please note that only switching events will influence the quality switching variance.

$$\sigma^2_{i,j} = \frac{\sum_{k=i+1}^{j} \Gamma_k \times (\theta_k \times t_k - \mu_{i,j} \times t_k)^2}{\sum_{k=i}^{j} t_k} \tag{3}$$

In order to consider the switching direction, i.e., up to a higher or down to lower quality level, we define ϕ in Equation 4 which returns 1 for up switches, -1 for down switches, and 0 in case no switch occurs.

$$\phi_k = \begin{cases} 1 & \text{if } \theta_k > \theta_{k-1} \\ -1 & \text{if } \theta_k < \theta_{k-1} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The *oscillation variance* ω in Equation 5 is similar to the quality switching variance and takes the direction of the quality switches into account. This metric converges to zero when an oscillation occurs and, thus, it shall be used in conjunction with the quality switching variance as shown in Section 4.

$$\omega^2_{i,j} = \frac{\sum_{k=i+1}^{j} \phi_k \times (\theta_k \times t_k - \mu_{i,j} \times t_k)^2}{\sum_{k=i}^{j} t_k} \tag{5}$$

3.2. Buffer-specific Metrics and Tools

In [4] we show that an adaptation algorithm based on a mathematical buffer model improves the efficiency of earlier adaptation algorithms [10] that were mainly based on throughput measurements and basic buffer metrics. The buffer model is based on a mathematical function that restricts the available quality levels based on the buffer fill state, which can be fitted to the available quality levels and the network conditions.

In this paper we provide a generic approach for describing this buffer model. We show examples for such models that calculate the quality level restriction ξ for a segment s_i based on the buffer fill state δ. Therefore, various functions may be used, e.g., linear (6a), exponential (6b), logarithmic (6c), etc. and parameterized with the variables as shown in Equation 6.

$$\xi(s_i) = a + \delta_i \times b \tag{6a}$$

$$\xi(s_i) = a \times e^{\delta_i \times b} \tag{6b}$$

$$\xi(s_i) = a \times log_b(\delta_i \times c) \tag{6c}$$

where $i \in [1, N]$ represents the segment index, δ_i the buffer fill state at decoding of segment i, and a, b, c are constant parameters to fine-tune the model.

The parameters can be used to increase or decrease the aggressiveness of the buffer model and as consequence also of the adaptation process. In this paper we adopt the logarithmic function, i.e., Equation 6c, which enables a smoother adaptation even for low quality levels as the slope of the function decreases monotonously. Thus, also low quality levels need a higher buffer fill states compared to the exponential approach. The exponential function, where the slope increases monotonously, is a more aggressive approach because lower quality levels can be already selected with very low buffer fill states. In [5] we used the exponential approach resulting in increased media throughput but also a higher risk for stalling.

Finally, Equation 7 calculates the worst case buffer $\delta_{wcb}(\theta_k)$ based on the bitrate of the segment θ_k and the segment length t_k as well as the minimum measured throughput of the current session min^j_i.

$$\delta_{wcb}(\theta_k)_{i,j} = \frac{\theta_k \times t_k}{min_{i,j}} \tag{7}$$

The output is the minimum buffer fill state in seconds that shall be available prior to the download of segment s_k which shall guarantee no stalls at the client while downloading this segment.

4. BUFFER-BASED ADAPTATION ALGORITHM WITH OSCILLATION DETECTION AND COMPENSATION

In this section we describe how to use the metrics and tools introduced in Section 3 to detect oscillations and to improve the adaptation logic compared to [4]. In particular, we combine the methods from [5] with [4] and apply a dynamic buffer model at runtime based on our metrics and tools. The improvements address the oscillation issues as described in Section 2, i.e., buffer overrun, competing bandwidth, and caching.

4.1. Oscillation Detection

For the oscillation detection we propose the *oscillation factor* ρ as defined in Equation 8 which depends on the quality switching variance σ and the oscillation variance ω over a given time window of segments.

$$\rho_{i,j} = \begin{cases} 1 - \frac{\omega_{i,j}}{\sigma_{i,j}} & \text{if } \sigma_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

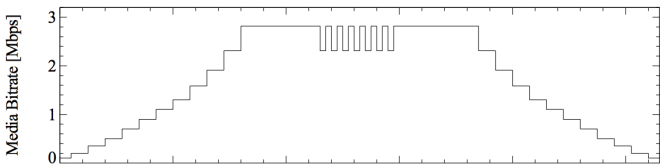
Figure 4 illustrates the oscillation detection based on a predefined adaptation behavior (the x-axis provides the seconds for all subfigures). Figure 4a shows a predefined adaptation behavior for a given media bitrate, which is used to demonstrate our metrics for a window size of 20 seconds. The quality switching variance (i.e., σ) and oscillation variance (i.e., ω) are shown in Figure 4b and Figure 4c depicts the oscillation factor (i.e., ρ).

At the beginning the client starts with the lowest available quality level and continuously switches up to the highest quality level. During that period both the quality switching variance and oscillation variance are equal, which results in a oscillation factor of zero. This behavior confirms our previous statement that the oscillation variance alone cannot be used to distinguish between plain quality switching and real oscillations. As soon as the the quality switching and oscillation variance start to become different, the oscillation factor increases which, in fact, indicates an oscillation as shown in Figure 4a around second 25.

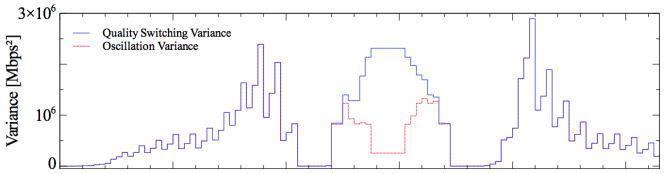
4.2. Buffer-based Adaptation

For our actual adaptation logic we adopt a logarithmic behavior (cf. Equation 6c) because the slope of the function decreases monotonously. This enables more flexibility in case of low buffer fill states, which means that the client can faster switch to higher quality levels at the beginning or recover from low buffer fill states, in comparison to an exponential behavior (cf. Equation 6b) where the slope increases monotonously. In the following section we will describe the semantics of the parameters a, b and c. We will also show that the parameter configuration works for arbitrary bitrates as we set parameter a depending on the maximum bitrate that is available in the manifest/MPD.

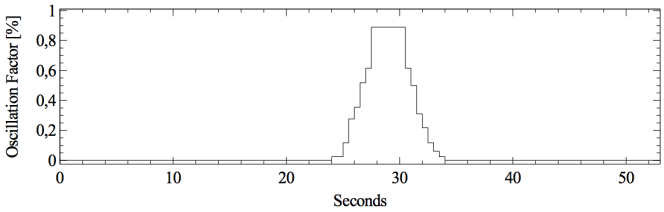
The parameter c can be used to enable a minimum buffer fill state referred to as steady state that the system should maintain before it is able to start the adaptation process, e.g., switching to a



(a) Predefined adaptation behavior for a given bitrate over time.



(b) Quality switching variance σ and oscillation variance ω which jointly indicate real oscillation if the values start to become different.



(c) Oscillation factor ρ which increases as soon as σ and ω start to become different.

Fig. 4: Oscillation Detection based on a predefined Adaptation Behavior.

different representation. This means that the client will download the lowest quality level until the buffer is at least filled with a certain amount of multimedia data until it starts to adapt to the current bandwidth conditions. In the following section we will show that only parameter c is influencing this steady state and that it can be used to modify the range of the steady state. We can use Equation 6c with $\xi = a \times log_b(\delta \times c)$ to show that parameter c sets the steady state of the buffer, i.e., the range where $\xi \leq 0$ and where $\xi > 0$, independent from the other two parameters, i.e., a and b.

For example, if we want our adaptation process to maintain a steady state until the buffer is filled at least 20%, we can use Equation 6c transformed, i.e., $\xi \leq 0 \rightarrow c \leq \frac{1}{\delta}$, with $\delta = 0.2$, which results in $c \leq 5$. Therefore, setting parameter c to 5 prevents the adaptation process until the buffer has reached a fill state of at least 20% ($\delta = 0.2$), because if $\xi \leq 0$, the lowest representation will be selected.

We have fitted parameter b while using parameter $c = 5$, which enables the steady state until the buffer is filled at least 20%. The buffer fill state ($0 \leq \delta \leq 1$) is ranging from 0 to 1; 0 means that the buffer is empty and 1 means that the buffer is full. Setting parameter c to 5 enables a range of 0 to 5 for the logarithmic function. Therefore, using a logarithm with the base of 5 ($b = 5$) will result in a maximum of 1, i.e., $\delta = 1$ and we define that the logarithm with $\delta = 0$ is equal to 0. Additionally, we set parameter a equal to the maximum available bitrate (according to the manifest/MPD), which enables an automatic adjustment to different media configurations based on the MPD. As a consequence, our parameter setup can be used for arbitrary bitrates. This setup would enable the maximum available bitrate only when the buffer is fully filled ($\delta = 1$) but for our adaptation process we decided to use a more aggressive setting which enables the maximum bitrate when the buffer is 80% filled. Therefore, we have used a base of 4, i.e., $b = 4$, which results in 1 when $\delta = 0.8$. In order to confirm the fitting above, we can

use Equation 6c to show if ξ is set to the maximum available media bitrate br_{max} and parameters a and c are used as above (i.e., a is br_{max} and $c = 5$) with $\delta = 0.8$ (i.e., 80% buffer fill state), that $br_{max} = br_{max} \times log_b(0.8 \times 5)$ results in $b = 4$. Furthermore, we define that $0 \leq \xi$. If the buffer is decreasing the client would also reduce the quality level, due to the fact that the buffer restriction would not allow high quality levels with low buffer fill states. Buffer underrun should not happen due to the adaptation behaviour but if it happens it would lead to stalls. The buffer will be measured in seconds and therefore an explicit relation between segments with size of seconds and buffer exists.

4.3. Compensation Algorithm

In order to reduce the oscillation, we use the oscillation detection as described in Section 4.1 in combination with a *Compensation Algorithm (CA)* defined in Algorithm 1. In case an oscillation is detected, the CA aims to smooth it. The CA will be activated when the oscillation factor exceeds a predefined $threshold$ (cf. lines 1-9) and has two phases: a low quality phase referred to as *low compensation* (cf. lines 10-26) and a high quality phase referred to as *high compensation* (cf. lines 19-26). In the low compensation, the adaptation logic maintains the low quality level of the detected oscillation phase until a backoff timer expires or when the buffer fill state falls below δ_{wcb} of the corresponding quality level. In the high compensation, it remains on the high quality level of the detected oscillation until the buffer fill state falls below the level as it was when the CA was activated. Additionally, the algorithm exits the high compensation when the buffer increases during that phase which indicates a bandwidth increase. This mechanism intends to smooth the oscillation utilizing the contents of the buffer.

```
1  if ρ_i^j > threshold and compensation == false then
2      saved_δ_i = δ_i;
3      low_quality = min_quality_i^j;
4      high_quality = max_quality_i^j;
5      compensation = true;
6      compensationLowQuality = true;
7      compensationHighQuality = false;
8      backoff = increase_backoff();
9  end
10 if compensationLowQuality == true then
11     decrease_backoff();
12     if backoff > 0 and δ_i < δ_wcb(low_quality_i^j) then
13         quality_level_next_segment = low_quality;
14     else
15         compensationLowQuality = false;
16         compensationHighQuality = true;
17     end
18 end
19 if compensationHighQuality == true then
20     if δ_i > saved_δ_i and δ_i decreasing then
21         quality_level_next_segment = high_quality;
22     else
23         compensationHighQuality = false;
24         compensation = false;
25     end
26 end
```

Algorithm 1: Compensation Algorithm (CA).

5. EXPERIMENTAL RESULTS

In this section we describe our experimental results. We compare our adaptation logic comprising the improved buffer model including the oscillation detection and compensation algorithm against a throughput-based adaptation logic based on the set-up as described in Section 2.

The results of our proposed adaptation logic without and with cache are shown in Figure 5 and Figure 6, respectively. Compared to the previous figures, they also include the oscillation factor and

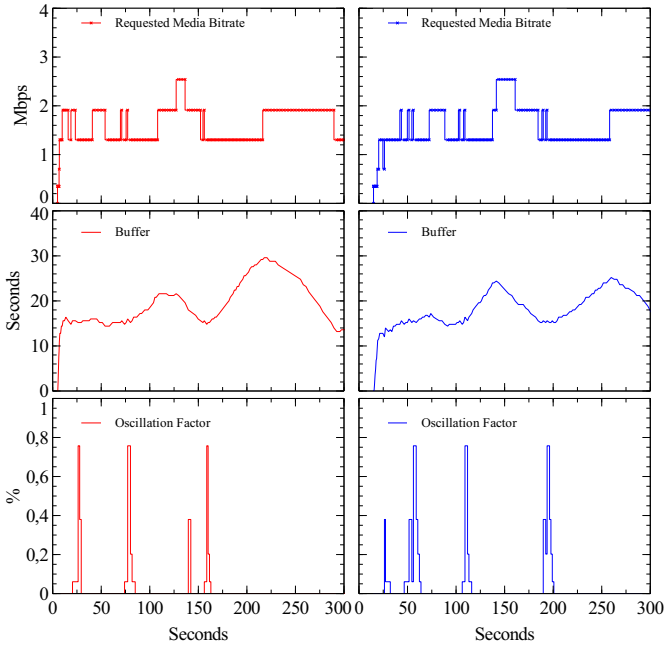


Fig. 5: Buffer Model based Adaptation including Compensation Algorithm without Cache for Client 1 (red/left) and Client 2 (blue/right).

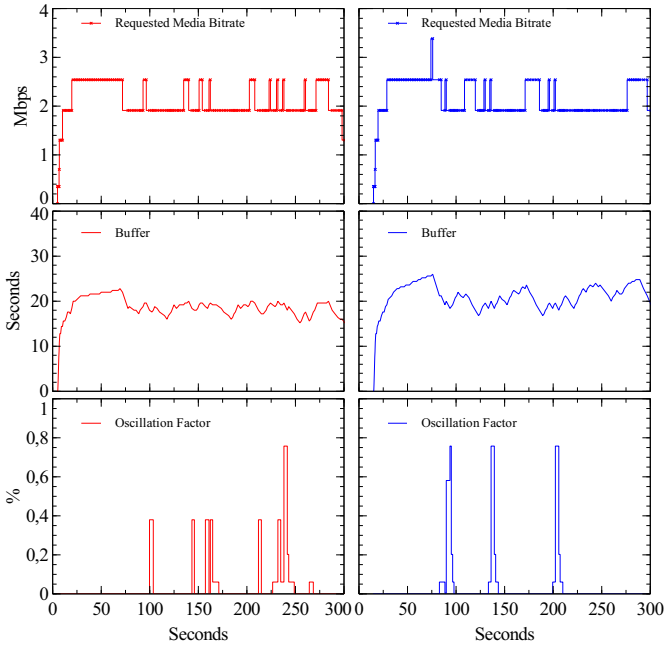


Fig. 6: Buffer Model based Adaptation including Compensation Algorithm with Cache for Client 1 (red/left) and Client 2 (blue/right).

the threshold is set to 0.7 (determined experimentally). Hence, each time the oscillation factor passes this threshold, the compensation algorithm is activated. Figure 5 shows the behavior of our buffer model based adaptation algorithm without a cache. The adaptation is much smoother compared to the throughput-based adaptation and after a stabilization phase at the beginning it never falls below the quality

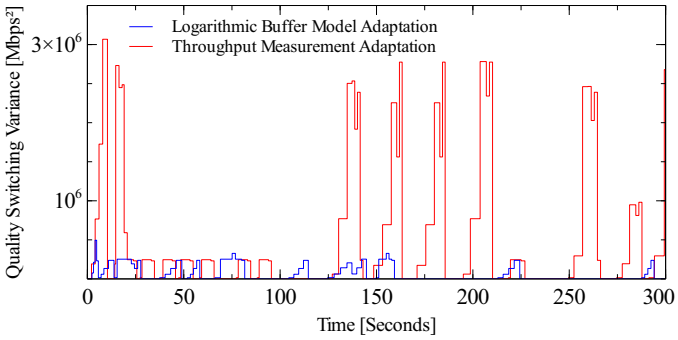


Fig. 7: Comparison of Quality Switching Variance for Client 1 without Cache.

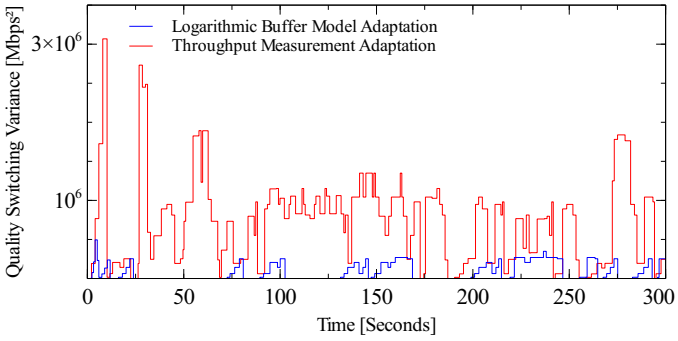


Fig. 8: Comparison of Quality Switching Variance for Client 1 with Cache.

level with 1300 kbps. Furthermore, the adaptation logic adequately detects oscillations and compensates those with our compensation algorithm. In particular, in second 25 the adaptation logic detects the oscillations and compensates it with the low quality level which lasts until approximately second 40 followed by the high compensation until the buffer reaches the previous state at around second 50. Another compensation phase starts at second 75 and the high compensation starts approximately at second 100, the point in time when the bottleneck increases to 3200 kbps. The increased bandwidth is recognized during the high compensation (second 125) which the adaptation logic preempts followed by a switch to a higher quality level. The adaptation logic detects another oscillation at second 150 which leads to another compensation phase that lasts almost until the end of the experiment.

We also evaluate our adaptation logic with a cache enabled and its results are shown in Figure 6. In comparison to the throughput-based adaptation it is able to utilize the cache in way that it can maintain a higher base quality (i.e., 1900 kbps) excluding the startup stabilization phase. Moreover, oscillations are detected, compensated, and the buffer fill state remains stable at an acceptable level.

Finally, we compare the average media bitrate and quality switching variance of our proposed adaptation logic versus the throughput-based adaptation logic. The results reveal that our proposed adaptation logic achieves for both clients on average a higher media bitrate of 11.2% if no caches are involved and 20% for the scenario with caches enabled. Concerning the quality switching variance, Figure 7 and Figure 8 show that our proposed adaptation logic has less quality switching variance, i.e., 86.95% for the scenario without cache and 90.75% if the cache is enabled.

6. CONCLUSIONS AND FUTURE WORK

In this paper we highlight problems in todays' adaptive streaming systems caused by throughput-based adaptation mechanisms and their influence when clients compete for bandwidth, specifically in the presence of caches. With the presented buffer-based adaptation models, clients metrics, and oscillation compensation algorithm we provide a comprehensive toolset to compensate for undesirable client behavior caused by incomplete network information and optimize the DASH streaming performance at the same time. The experimental results validate our findings, i.e., oscillation detection and compensation (with and without caches enabled) while increasing the media throughput at the client (+11.2% without cache, +20% with cache) and increasing the hit rate at the cache (+17.42%) while maintaining a steady buffer fill state. Finally, our oscillation detection and compensation algorithm is a client-centric approach which enables scalability and keeps maintaining the advantages of DASH.

Future work items comprise large-scale evaluations within real-world environments using actual bandwidth traces and heterogeneous client implementations, e.g., a mixture of (old-fashioned) progressive download clients and DASH-based clients with different adaptation logics including those which are specifically addressing similar issues as addressed in this paper (e.g., [7]). Finally, the parameters for the adaptation logic could be fine-tuned and validated through (crowdsourced) subjective quality assessments.

7. REFERENCES

[1] Iraj Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[2] Saamer Akhshabi, Sethumadhavan Narayanaswamy, Ali C. Begen, and Constantine Dovrolis, "An Experimental Evaluation of Rate-Adaptive Video Players over HTTP," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 271 – 287, 2012.

[3] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, New York, NY, USA, 2011, MMSys '11, pp. 169–174, ACM.

[4] C. Mueller, D. Renzi, S. Lederer, S. Battista, and C. Timmerer, "Using Scalable Video Coding for Dynamic Adaptive Streaming over HTTP in Mobile Environments," in *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, 2012, pp. 2208–2212.

[5] C. Mueller, S. Lederer, and C. Timmerer, "A Proxy Effect Analysis and Fair Adaptation Algorithm for Multiple Competing Dynamic Adaptive Streaming over HTTP Clients," in *Visual Communications and Image Processing (VCIP), 2012 IEEE*, 2012, pp. 1–6.

[6] Rémi Houdaille and Stéphane Gouache, "Shaping HTTP Adaptive Streams for a Better User Experience," in *Proceedings of the 3rd Multimedia Systems Conference*, New York, NY, USA, 2012, MMSys '12, pp. 1–9, ACM.

[7] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen, "Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players," pp. 19–24, 2013.

[8] Wei Pu, Zixuan Zou, and Chang Wen Chen, "Video Adaptation Proxy for Wireless Dynamic Adaptive Streaming over HTTP," in *Packet Video Workshop (PV), 2012 19th International*, 2012, pp. 65–70.

[9] Ali El Essaili, Damien Schroeder, Dirk Staehle, Mohammed Shehada, Wolfgang Kellerer, and Eckehard Steinbach, "Quality-of-Experience driven Adaptive HTTP Media Delivery," in *IEEE International Conference on Communications (ICC 2013)*, Budapest, Hungary, Jun 2013.

[10] Christopher Mueller, Stefan Lederer, and Christian Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments," in *Proceedings of the 4th Workshop on Mobile Video*, New York, NY, USA, 2012, MoVid '12, pp. 37–42, ACM.