Online learning for low-latency adaptive streaming

Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, Arjen Wagenaar
firstname@unified-streaming.com
Unified Streaming
Amsterdam, The Netherlands

ABSTRACT

Achieving low-latency is paramount for live streaming scenarios, that are now-days becoming increasingly popular. In this paper, we propose a novel algorithm for bitrate adaptation in HTTP Adaptive Streaming (HAS), based on Online Convex Optimization (OCO). The proposed algorithm, named *Learn2Adapt-LowLatency (L2A-LL)*, is shown to provide a *robust* adaptation strategy which, unlike most of the state-of-the-art techniques, does not require parameter tuning, channel model assumptions, throughput estimation or application-specific adjustments. These properties make it very suitable for users who typically experience fast variations in channel characteristics. The proposed algorithm has been implemented in DASH-IF's reference video player (dash.js) and has been made publicly available for research purposes at [22]. Real experiments show that *L2A-LL* reduces latency significantly, while providing a high average streaming bit-rate, without impairing the overall Quality of Experience (QoE); a result that is independent of the channel and application scenarios. The presented optimization framework, is robust due to its design principle; its ability to learn and allows for modular QoE prioritization, while it facilitates easy adjustments to consider applications beyond live streaming and/or multiple user classes.

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

KEYWORDS

Adaptive video streaming, low latency, online Optimization

ACM Reference Format:
Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, Arjen Wagenaar. 2020. Online learning for low-latency adaptive streaming. In *11th ACM Multimedia Systems Conference (MMSys'20), June 8–11, 2020, Istanbul, Turkey*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3339825.3397042

1 INTRODUCTION

Video streaming accounts nowadays for more than 60% of the global Internet traffic, a percentage projected to reach a striking 78% by 2021 [6]. To facilitate this increasing multimedia consumption demand, a decade ago the industry transitioned to the HAS principle for video delivery. Most notably the MPEG Dynamic Adaptive Streaming over HTTP (DASH) ISO standard [10], along with IETF's HTTP Live Streaming (HLS) [18] specification, have been adopted as the most widespread video delivery methods, allowing Content Delivery Networks (CDN) to leverage the full capacity of existing Hypertext Transfer Protocol (HTTP) infrastructure; instead of relying upon networks of dedicated streaming servers.

These HAS services introduced settings, such as partitioning the video sequence in smaller fragments of constant duration at the server and pre-buffering at the client, that have a direct impact on the QoE [19] and achievable Quality of Service (QoS) [13]. In streaming, timeliness of video delivery is measured by latency; the time difference between the moment video content is generated and the moment it is rendered on the client's screen. Until recently latency in the regime of > 30 s has been the norm, associating Over-The-Top (OTT) delivery withg latency higher than broadcast.

A first approach to tackle this discrepancy and to reduce end-to-end latency, to at least match broadcast standards, came in the form of shortening the segment duration. However, such an approach can diminish visual quality, and often small segment durations may lead to unstable delivery infrastructure in CDNs.

Another approach that has now been adopted as the main discipline for low-latency streaming is a combination of a new container format and the chunked transfer property supported natively in HTTP 1.1 and beyond. Common Media Application Format (CMAF) [11] is simply a standardized container that can hold video, audio, or text data. The efficiency of CMAF, that is deployed using either HLS or DASH, is driven by the fact that CMAF-wrapped media segments can be simultaneously referenced by HLS playlists and DASH manifests. This enables content owners to package and store one set of files, halving storage costs. In Section 2 we provide a detailed specification of how media are organized according to CMAF in multiple quality presentations, i.e. multiple resolutions to meet the diverse display capabilities of different types of user devices and multiple encoding rates to facilitate adaptation to changing network characteristics. Further, each quality presentation is partitioned in multiple fragments (or chunks) to increase the granularity of the bitrate adaptation decisions and to reduce latency. When coupled with chunked transfer, all the means necessary to significantly reduce live streaming latency are in place.

Nonetheless, although chunked CMAF transfer has set the stage for latency reduction, optimal bitrate adaptation over fluctuating channels remains an elusive task. Moreover, low-latency constraints pose additional strain on bitrate adaptation algorithms. The application-level queue at the video client used for storing downloaded parts of the video, also known as the buffer, is deployed to protect the client from abrupt changes in the communication channel (throughput, jitter etc.). By deduction, reduction in buffer service-time equals reduction in latency. Therefore in low-latency

MMSys'20, June 8–11, 2020, Istanbul, Turkey

ACM ISBN 978-1-4503-6845-2/20/06…$15.00
https://doi.org/10.1145/3339825.3397042

applications, the buffer length is typically constrained an upper limit (buffer target value), in the order of the aimed latency. A shorter buffer though, offers less protection against channel state estimation errors, propagated to the bitrate decisions, which in turn can have a detrimental effect on streaming experience. Thus, one of the main challenges faced by the multimedia research community currently, concerns accurate throughput estimation to support bitrate adaptation, particularly in the context of low-latency streaming.

The scope of this paper is to offer a novel perspective on the problem, from the point of view of OCO, mitigating the requirement for throughput estimation altogether. *A low-latency bitrate adaptation algorithm is, in essence, an optimization solution with the objective of minimizing latency, while at the same time maximizing achievable video bitrate and ensuring uninterrupted streaming.*

Primarily, heuristic approaches have been proposed for bitrate adaptation, that are mainly classified into two main categories according to the input dynamic considered for adaptation. First, throughput-based methods [4, 14] estimate the available channel rate to decide on the bitrate of the streamed video. These methods are as accurate as their throughput estimation module. Second, buffer-based methods [9, 21] use application level signals, such as the instantaneous buffer level to perform the adaptation. Such methods become highly unstable when used in the very small buffer regime of low-latency [12].

Recent adaptation algorithms, in an effort to overcome these limitations, resort to learning techniques such as reinforcement learning [23] or dynamic programming [24] to attain optimal quality adaptation. However, their practical implementation for low-latency may be hindered by the complexity of exploring the complete optimization space or by channel model requirements.

OCO has emerged [25] as a very effective online learning framework, that is also lightweight in terms of resource requirements and computation. OCO is an online optimization method, where an agent learns to make sequential decisions in order to minimize an adversarial loss function, unknown at decision time. OCO is "model-free" , as no assumption is required to be made for the statistical model of the channel, while at the same time it provides tractable feasibility and performance guarantees [3]. Having already been proposed for problems such as cloud resource reservation management [16] and dynamic resource network allocation [5], it represents an appealing candidate for optimization of low-latency bitrate adaptation as well. However, the actual application of OCO in bitrate adaptation is not a straightforward task. Given that bitrate adaptation has a discrete decision space (set of available quality presentations) and instantaneous state-dependent constraints (finite-sized buffer queue), it does not fall directly in the class of OCO problems.

This work provides multiple contributions towards formulating the bitrate adaptation optimization problem under the OCO framework. First, we model the adaptive streaming client by a learning agent, whose objective is to minimize the average buffer displacement (latency) of a streaming session. Second, we fulfill the OCO requirement that both the set of decisions and constraint functions must be convex by a) allowing the agent to make decisions on the video quality of each chunk, according to a probability distribution and by b) deriving a convex constraint associated with the upper bound of the finite buffer queue. We achieve the latter by making a relaxation to an unbounded buffer that adheres to time-averaging constraints. Third, we model the channel rate evolution by an adversary, that decides the cost of each decision only after it has been taken. *Thus all bitrate decision are based on historical values without requiring any throughput estimations.* We eventually solve the bitrate adaptation optimization problem by proposing *Learn2Adapt-LowLatency* (*L2A-LL*), a novel, online-learning, adaptation algorithm based on the OCO theory. One of the strong properties of *L2A-LL*, is that its formulation is modular, allowing to incorporate more QoE factors (and their weight prioritization) to account for different QoE objectives and for multiple streaming scenarios and/or user classes. Additionally, *L2A-LL* provides performance guarantees, that can be very useful for commercial deployment, while it does not require any statistical assumption for the unknowns or estimations. These properties make *L2A-LL* a robust bitrate adaptation solution.

L2A-LL has been implemented in the DASH-Industry Forum's reference player (dash.js [1]) and is publicly available at [22].

2 SYSTEM MODEL

This section introduces the model for the media content and client operations used in the rest of this work.

In live streaming scenarios, reduced end-to-end latency is a hard constraint. To perform low-latency at scale, unlike segment-level pull-based approaches, such as DASH or HLS, chunked encoding is required. Chunked transfer encoding is a data transfer mechanism, available since version 1.1 of the HTTP, in which the server does not know the final size of the object it is sending. As a consequence, it does not insert a content-length header in the response and instead sends data as it becomes available in short bursts, called 'HTTP chunks'. The transmission ends when a zero-length chunk is sent, and the receiver closes the connection. Per the MPEG CMAF standard [11], a CMAF track is composed of a number of objects. A 'CMAF chunk' (not to be confused with an HTTP chunk) is the smallest reference-able unit, while one or more chunks are combined to form a fragment, and one or more fragments are combined to form a segment. Typically each chunk is encoded at N quality presentations (CMAF Presentation) at increasing *target encoding bit-rate* $\mathcal{R} = (R_1, \dots R_N)$.

In the following, we refer to the, typically variable, interval between two consecutive chunk downloads as a *decision epoch* $t \in 1, \dots, T$, where T is the total number of decision epochs, referred to as *the horizon*. In VoD streaming applications T is known and computed as $T = \lceil D/V \rceil$, where D is the video sequence duration and V the chunk duration (typically constant for the duration of the stream). For live streaming applications, D is unknown and thus T is selected according to the optimization convergence horizon. [1]

At the beginning of the t-th epoch, the client requests the CMAF Presentation $x_t \in \mathcal{X} = \{1, \dots, N\}$ for chunk t, corresponding to target encoding bit-rate $r_{x_t} \in \mathcal{R}$. According to the adaptive streaming discipline, the client can affect the video streaming with its decisions $x_t, \forall t = 1, \dots, T$. For instance, although choosing r_N

[1] While some low-latency streaming profiles also support chunks that do not provide, a switching or decision opportunity at the boundary, without loss of generality, we assume switching at the chunk boundaries is possible. Popular streaming profiles like [8] or [7] also support this explicitly by allowing short fragments composed of a single chunk.

all the time produces high video quality, it imposes the downloading of a large volume of data, which the available channel might not support. Therefore, the idea is to adjust the decisions to as much $r_n \in \mathcal{R}$ as the channel can support, an objective that also provisions against increasing the overall latency.

For a given presentation $x_t \in \mathcal{X} = \{1, \ldots, N\}$, the *actual size* of the t-th chunk ($t \in [1, \ldots, T]$) – denoted $S_{t,n}$ and measured in bits – is a function of content. High-motion scenes would require more bits to be encoded at the same quality presentation, compared to low-motion scenes. Without loss of generality, as data size information is not included in the manifest files, following spatiotemporal correlation models for typical video sequences [19], data sizes are approximated as $S_{t,n} \approx V \cdot r_n$.

The downloaded chunks are stored in a buffer whose size may only temporarily exceed an upper bound B_{max}, e.g. due to memory constraints of the client device. The length of the buffer is directly correlated to the latency imposed in the stream, as larger queues are associated with longer service-times (playback). Thus in general, for low-latency applications, buffer lengths are targeted at lower values B_{target}. Let B_t represent the buffer level at the beginning of the t-th epoch, measured in seconds of video. Upon completely downloading the t-th chunk, B_t increases by V seconds (same for all chunks). Due to playback, B_t is also consumed at playback rate p_t, which for general intents and purposes is equal to a unit rate (as long as $B_t > 0$). For time-sensitive streaming applications, a common strategy to reduce latency is to allow the client to alter p_t, in order to catch up with the live edge.

In the following, we will assume that the server is connected to the client across a downlink channel of average rate C_t. Namely, C_t is the average effective downlink rate measured for the total period that the client downloads the t-th chunk.

Since the epoch duration is equal to the chunk size over the channel rate ($S_{t,x_t} \approx V * R_{x_t})/C_t$, the buffer evolves according to:

$$B_{t+1} = \left[B_t - \frac{p_t \cdot S_{t,x_t}}{C_t}\right]^+ + V - \Delta_t, \tag{1}$$

where $[x]^+ \triangleq \max(0, x)$. To account for the upper bound B_{max} of the buffer, a delay $\Delta_t = \left[\left[B_t - \frac{p_t \cdot S_{t,x_t}}{C_t}\right]^+ + V - B_{max}\right]^+$ is introduced. In other words, if $\left[B_t - \frac{p_t * S_{t,x_t}}{C_t}\right]^+ + V < B_{max}$, the $(t+1)$-th chunk is downloaded immediately and $\Delta_t = 0$. Otherwise, the $(t+1)$-th chunk is delayed by Δ_t seconds, to allow the buffer to drop below B_{max}. This delay protects against *buffer overflow* incidents, which occur when the buffer surpasses B_{max}. Typically in low-latency streaming mode, the target buffer level $B_{target} \ll B_{max}$, thus Δ_t is rarely imposed.

End-to-end latency is defined as the time difference between the moment video content is generated and the moment it is played-out on the client's screen. An adaptive bitrate algorithm, can only control the time required for the content to be transmitted over a variable and unknown at decision time channel rate. Thus in the context of this work, latency (the relevant counter-part to an adaptive streaming algorithm) is associated with the buffer length. Keeping the latency low, means not allowing the buffer to grow, while provisioning that it stays in the positive regime. It is worth mentioning that a variable playback rate can also affect latency, yet this is a separate client operation and not part of the bitrate adaptation algorithm. Besides, the impact of a variable playback rate on QoE has not been well documented [19] and is thus left outside our main optimization scope, while it can be added later.

Table 1: Notations

Notation	Definition	Units
V	chunk duration	seconds
T	Optimization horizon	scalar
N	Quality presentations	scalar
x_t	Selected quality presentation for chunk t	scalar
r_{x_t}	Bitrate corresponding to quality x_t	kbps
$S_{t,n}$	File size of chunk t in n-th quality	kbits
ω_t	Decision distribution	probability vector
ω^*	Benchmark distribution	probability vector
Q	Virtual queue	scalar
V_L	Cautiousness parameter	scalar
α	Step-size	scalar
C_t	Measured channel rate at epoch t	kbps
B_t	Buffer level at epoch t	seconds
B_{max}	Maximum buffer level	seconds
B_{target}	Target buffer level (latency)	seconds
Δ_t	Buffer delay	seconds

A *buffer underflow* occurs when the instantaneous buffer level drops below zero, causing a *stall* in the video playback, an event that significantly degrades the QoE. In the next section we provide a machine learning framework which allows us to design a simple learning algorithm that provably optimizes latency subject to keeping the buffer asymptotically away from the two limits. The goal of the framework we develop in the following is to strike a favorable trade-off between minimizing the possibility of a buffer underflow (or overflow) and minimizing the latency, under no statistical assumption of the operation network.

3 BITRATE ADAPTATION VIA ONLINE CONVEX OPTIMIZATION (OCO)

This section provides an algorithmic solution based on the theory of OCO. In particular in the following, we cast the low-latency bitrate adaptation decision as an OCO with budget constraints problem and in the process, we introduce all necessary modelling assumptions (utility and constraint functions) along with a set of relaxations that are required by the OCO framework. Last, we present our online-learning algorithm *Learn2Adapt-LowLatency* (*L2A-LL*), based on the gradient descent operation, and we invoke a theorem from [15] that provides theoretical guarantees for its performance.

3.1 Problem formulation

Our first step is to define the bitrate adaptation problem as a *constrained OCO*, where the goal is to minimize the cumulative losses $\sum_{t=1}^{T} f_t(x_t)$ (referring to latency) while keeping the cumulative constraint function $\sum_{t=1}^{T} g_t(x_t)$, negative (referring to buffer underflow); see also the relevant literature [15, 17]. In the OCO framework, functions f_t, g_t are chosen by an *adversary* and are unknown at decision time. We will relate these functions to the random evolution of the channel rate C_t, which is not adversarial. Nevertheless, the adversary setting is general enough to include any – potentially time-varying – distribution of C_t. In general it is worth mentioning

that the adversarial setting is general enough to include all dynamics of the channel and thus it mitigates throughput estimation requirements; a property that facilitates robust bitrate adaptation.

Recall the set of quality presentation $\mathcal{X}$, and let $x_t \in \mathcal{X}$ be the decision for the chunk to be downloaded in epoch t. Consider the following functions:

$$\tilde{f}_t(x_t) := V - \frac{V \cdot R_{x_t}}{C_t} \tag{2}$$

$$\tilde{g}_t(x_t) := \frac{V \cdot R_{x_t}}{C_t} - V, \tag{3}$$

where (2) expresses the penalty (added latency at every epoch) for selecting the corresponding encoding bitrate level (high bitrate yields smaller losses), and (3) constraints the encoding decisions in the positive latency regime and also expresses the buffer displacement, which will be used to model the buffer underflow constraint. A high bitrate R_{x_t} combined with a low channel rate C_t will prolong download time $\frac{V \cdot r_{x_t}}{C_t}$, which results in higher latency and buffer consumption. In general, we will aim at keeping the total latency low ($min \sum_{t=1}^{T} \tilde{f}_t(x_t)$ but constraint it in the positive regime ($\sum_{t=1}^{T} g_t(x_t) \leq 0$), on average. Since C_t is unknown at decision time of x_t, it is impossible to know the values of $\tilde{g}_t(x_t)$. Our approach therefore, is to learn the best x_t based on our estimation of $\tilde{g}_t(x_t)$.

Yet, in order to formally cast the above problem as an OCO with budget constraints, the following complications need to be resolved before. First, we provide a relaxation to the hard constraint of the buffer model ($B_t \in [0, B_{max}]$, a non-continuous function). Second, since the set of representations is discrete and therefore not convex, the OCO framework requires a convexification operation of the decision set by means of randomization. In this direction by associating a probability to each decision, we learn the probability distribution of the decisions.

3.1.1 ***Buffer constraints.*** Here we propose a relaxation on the finite buffer queue, to be inline with OCO's requirement for convex constraints.

First we explain how we use the cumulative constraint function $\sum_{t=1}^{T} \tilde{g}_t(x_t)$ to model buffer underflow. Recall that the buffer evolves according to (1) and notice that ensuring $0 \leq B_t \leq B_{max}$, $\forall t$, is in principle a very complicated control problem, which in the presence of unknown adversarial C_t is exacerbated.

To avoid computationally heavy approaches (such as reinforcement learning) and to arrive at a simple (yet robust) solution, we seek an alternative approach. In that direction, we treat the buffer as an infinite queue, with the simpler (compared to (1)) update rule: $B_{t+1} = B_t + V - p_t \cdot V \cdot r_{x_t}/C_t$. By this, we allow instantaneous violation of the buffer budget, which incurs a penalty according to the queue's deviation ($\tilde{g}_t(x_t)$). This penalty method will maintain the buffer on the positive regime on average.

Using (3), we capture in $\tilde{g}_t(x_t)$ the instantaneous buffer displacement (measured in seconds) and by requiring the cumulative constraint $\sum_{t=1}^{T} \tilde{g}_t(x_t) \leq 0$, we ensure that on average B_t remains in the non-negative regime. At the same time, our objective to minimize instantaneous latency would lead to bitrate decisions that, on average, keep the buffer below the $B_{target} \ll B_{max}$ range. Thus we ensure, on average, both a lower and an upper bound on the, now, infinite buffer queue.

A benefit is that this constraint is in the realm of OCO theory, and therefore it allows us to design a simple learning algorithm that provably satisfies it. Another benefit, is that in the unfortunate event of a stall, the negative part of the above expression is "used" as a protection cushion, which makes future under-flows more improbable. Overall, our approach here is to apply a loosely coupled control to the buffer constraint, by tolerating instantaneous violations and ensuring that in the long-term only a few are experienced.

3.1.2 ***Convexification.*** To obtain a convex decision set, we use a convexification operation based on randomization of the decision process [20]. Consider the probability simplex:

$$\Omega = \{\boldsymbol{\omega} \in \mathbb{R}^N : \boldsymbol{\omega} \geq 0 \wedge \|\boldsymbol{\omega}\|_1 = 1\},$$

where $\omega_i = \mathbb{P}(x = i)$ denotes the probability that we decide $x = i$, $i \in \mathcal{X}$. Ω is a convex set. Instead of learning to decide x_t (which is selected from an integer set), our approach is to learn to decide a probability distribution $\boldsymbol{\omega}_t = (\omega_{t,i})_{i=1,\ldots,N}$ selected from Ω. Given a decision $\boldsymbol{\omega}_t$, the actual encoding level will be chosen according to the computation of its expectation. The functions of interest become now random processes and we must appropriately modify them by taking expectations (with respect only to $\boldsymbol{\omega}_t$ and not the randomness of C_t):

$$f_t(\boldsymbol{\omega}_t) := V - \frac{V * \mathbb{E}[R_{x_t}]}{C_t} = V - \frac{V * \sum_i \omega_{t,i} R_i}{C_t} \tag{4}$$

$$g_t(\boldsymbol{\omega}_t) := \frac{V * \mathbb{E}[R_{x_t}]}{C_t} - V = \frac{V * \sum_i \omega_{t,i} R_i}{C_t} - V \tag{5}$$

Given the loss function and constraints above we formulate the constrained OCO problem, that we solve in Section 4:

$$\min_{\boldsymbol{\omega} \in \Omega} \sum_{t=1}^{T} f_t(\boldsymbol{\omega}) \quad \text{s.t.} \quad \sum_{t=1}^{T} g_t(\boldsymbol{\omega}) \leq 0. \tag{6}$$

3.1.3 ***Performance metric.*** We now define the performance metric in our problem which consists of two parts: the *regret* of an algorithm and the constraint residual, defined as:

$$R_T = \sum_{t=1}^{T} f_t(\boldsymbol{\omega}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{\omega}^*) \quad \text{and} \quad V_T = \sum_{t=1}^{T} g_t(\boldsymbol{\omega}_t), \tag{7}$$

respectively. Here $\boldsymbol{\omega}^* \in \Omega$ is a benchmark distribution, that minimizes the losses when the entire sample path and the functions f_t, g_t are known, while additionally ensuring the cumulative constraints every K slots:

$$\boldsymbol{\omega}^* \in \arg\min_{\boldsymbol{\omega} \in \Omega} \sum_{t=1}^{T} f_t(\boldsymbol{\omega}) \text{ s.t. } \sum_{t=k}^{K+k-1} g_t(\boldsymbol{\omega}) \leq 0, \quad \forall k = 1, \ldots, T - K + 1.$$

As [15] explains, any $K = o(T)$ suffices to define a benchmark that can be learned and in particular taking $K = T^{1-\epsilon}$, for small $\epsilon > 0$, we ensure that the used benchmark is constrained in almost the same manner as the ideal benchmark.

If an algorithm achieves both $o(T)$ regret and $o(T)$ constraint residual, then it follows that as $T \to \infty$ we have (i) $R_T/T \to 0$, hence our algorithm has the same losses with (or "learns") the benchmark action, and (ii) $V_T/T \to 0$, hence our algorithm ensures the average constraint. As we will see, the benchmark action is the

best *a posteriori* action, taken with knowledge of all the revealed values of C_t, and therefore learning it is both remarkable and useful.

4 LEARN2ADAPT-LOWLATENCY SOLUTION

In this section, we start from the constrained OCO problem defined in (6), and use it to design a "no regret" algorithm. We first provide the intuition behind the design, then explain the algorithm, and finally provide theoretical performance guarantees.

As a general note, a main challenge in such problems is that the function $g_t(\boldsymbol{\omega}_t)$ is not known at decision time of $\boldsymbol{\omega}_t$, and the idea in OCO is to predict these functions using a first order Taylor expansion of g_{t-1} around $\boldsymbol{\omega}_{t-1}$ evaluated at $\boldsymbol{\omega}_t$ [25]:

$$\hat{g}_t(\boldsymbol{\omega}_t) := g_{t-1}(\boldsymbol{\omega}_{t-1}) + \langle \nabla g_{t-1}(\boldsymbol{\omega}_{t-1}), \boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1} \rangle. \quad (8)$$

In (8), $\boldsymbol{\omega}_t$ is the only value to be determined at t, whereas historical quantities $\boldsymbol{\omega}_{t-1}$, $\nabla g_{t-1}(\boldsymbol{\omega}_{t-1})$ and $g_{t-1}(\boldsymbol{\omega}_{t-1})$ are known via the obtained feedback. In other words, the bitrate decision is solely based on historical values, thus *effectively* mitigating the requirement for throughput estimation.

Contrary to the standard (unconstrained) online gradient [25], however, our algorithm must combine the objective and the constraint functions. To this end, consider the regularized Lagrangian:

$$L_t(\boldsymbol{\omega}, Q(t)) = Q(t)\hat{g}_t(\boldsymbol{\omega}) + V_L \hat{f}_t(\boldsymbol{\omega}) + \alpha ||\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1}||^2, \quad (9)$$

where $Q(t)$ is a Lagrange multiplier, $\hat{g}_t(\boldsymbol{\omega})$ is the prediction of the constraint function $g_t(\boldsymbol{\omega})$ from (8), V_L is a cautiousness parameter, $\hat{f}_t(\boldsymbol{\omega})$ applies (8) to f_t, α is a step-size, and $||\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1}||^2$ is a regularizer that smoothens the decisions. The addition of the regularizer provides modular incorporation of supplementary QoE provisions in our framework. Parameters V_L and α are tuned for convergence and their choices are given below. We mention here, that the Lagrange multiplier is updated in a *dual ascent* approach, by accumulating the constraint deviations:

$$Q(t+1) = [Q(t) + \hat{g}_t(\boldsymbol{\omega}_t)]^+. \quad (10)$$

The following algorithm takes a step in the direction of the sub-gradient of the regularized Lagrangian (9):

Algorithm 1 Learn2Adapt-LowLatency (L2A-LL)

Initialize: $Q(1) = 0$, $\boldsymbol{\omega}_0 \in S$
Parameters: cautiousness parameter V_L, step size α
1: **for** all $t \in \{1, 2, \ldots, T\}$ **do**
2: $\quad \boldsymbol{\omega}_t = \text{proj}_\Omega \left[\boldsymbol{\omega}_{t-1} - \frac{V_L \nabla f_{t-1}(\boldsymbol{\omega}_{t-1}) + Q(t) \nabla g_{t-1}(\boldsymbol{\omega}_{t-1})}{2\alpha} \right]$
3: $\quad Q(t+1) = [Q(t) + \hat{g}_t(\boldsymbol{\omega}_t)]^+$
4: **end for**

Here $\text{proj}_\Omega[\cdot]$ denotes the Euclidean projection on set Ω. We have the following performance guarantees for our algorithm.

At every decision epoch $t = 1, 2, \ldots T$ the following events occur in succession:

(a) the learning agent computes $\boldsymbol{\omega}_t \in \Omega$ according *L2A-LL*
(b) the agent chooses $x_t \in \arg\min_{x \in \mathcal{X}} |R_x - \sum_{n=1}^{N} \omega_{t,n} R_n|$,
(c) an adversary decides C_t, and the loss function $\tilde{f}_t(\boldsymbol{\omega}_t)$ and the constraint function $\tilde{g}_t(\boldsymbol{\omega}_t)$ are determined using (2)-(3), and then used to measure the actual loss and buffer displacement,
(d) one of the following forms of feedback are provided to the agent: (i) the value of C_t, (ii) the form of f_t, g_t, (iii) the value of gradients $\nabla f_t(\boldsymbol{\omega}_t), \nabla g(\boldsymbol{\omega}_t)$.

The feedback above is used by the agent to eventually determine the gradient vectors $\nabla f_t(\boldsymbol{\omega}_t), \nabla g_t(\boldsymbol{\omega}_t)$.

Based on the defined optimization performance metric (7), *L2A-LL* enjoys the following performance guarantees:

Theorem 4.1 (From [15]). *Choose small $\epsilon > 0$, fix $K = o(T^{1-\epsilon})$, $V_L = T^{1-\epsilon/2}$, and $\alpha = V_L\sqrt{T}$. Then, the* L2A-LL *algorithm guarantees:*

$$R_T = O(T^{1-\epsilon/2}), \qquad V_T = O(T^{1-\epsilon/4}).$$

Effectively, this means that over time our algorithm learns the best a-posteriori distribution $\boldsymbol{\omega}^*$, which neatly satisfies the average constraints and minimizes the cumulative latency. We show below that the corresponding choices x_t made by sampling this distribution have extremely well performing properties for video streaming adaptation.

5 EXPERIMENTAL EVALUATION

Learn2Adapt-LowLatency (L2A-LL) has been implemented in 'dash.js' [1], DASH-IF's reference video player and is publicly available for experimentation at [22]. According to the requirements of the Multimedia Systems (MMSys) 2020 Grand Challenge on Adaptation Algorithms for Near-Second Latency Bitrate Adaptation, we have followed the test environment implementation guidelines [2].

5.1 Experimental setup

Our open source software package [22], allows the interested reader to download, run and test our implementation that includes the modified dash.js instance (to include *L2A-LL*) and the required server and orchestration nodes. We have evaluated *L2A-LL* implementation in dash.js' low-latency mode for a set of experiments that rely on 5 different network profiles and a test video sequence (encoded at $\mathcal{R} = 0.3, 0.6, 1.0$ Mbps) and organized in chunks with duration $V = 0.5$ s), both provided by the organizers of the challenge. For the purposes of experimentation we have allowed a variable playback rate p_t but left its selection to the dash.js player. As optimization horizon we have selected a small value of $T = 4$, to account for the time-sensitive nature of low-latency streaming, while for the convergence parameters α and V_L we followed the recommendations of Theorem 4.1. For the specification of the network profiles used in the evaluation below (Cascade, Intra-Cascade, Spike, Slow jitters and Fast jitters) we refer to [2]. The implemented instance of *L2A-LL* in dash.js exists in *'Learn2Adapt-LowLatency/dash.js/src/streaming/rules/abr/L2ARule.js'* of [22], while *'Learn2Adapt-LowLatency/dash.js/samples/low-latency/index.html'* holds all required player configuration.

5.2 Experimental results

Table 2 provides results for an experimental evaluation that is based on QoE metrics such as: average bitrate, average latency, stall duration, average buffer length and number of switches. Additionally

Table 2: Experimental results for L2A-LL

Network profile	Regret/T	Constraint residual/T	Avg Bitrate (Mbps)	Avg Buffer length (s)	Latency (s)	Stall duration (s)	Num. Switches
Cascade	0.02	0.008	0.58	0.46	1.7	28	10
Intra-Cascade	0.05	0.007	0.35	0.4	2.5	38	7
Spike	0.008	0.001	0.59	0.4	1.5	7	3
Slow jitter.	0.038	0.005	0.33	0.47	1.3	7	4
Fast jitter	0.04	0	0.33	0.57	1.2	0.7	2

evaluate *L2A-LL*'s performance on the regret and constraint residual metrics (7) against the benchmark distribution $\omega*$ defined in Section 3.1.3. This comparison aims at showing the benefit of an optimized bitrate adaptation algorithm in terms of latency, that also respects the theoretical performance guarantees. The latter is shown the by the close-to-zero values that both the regret *rate* and the constraint residual *rate* obtain in all evaluated network profiles.

Through these preliminary results, we can infer that *L2A-LL* manages to achieve low latency values, while overall keeping a high average bitrate. The algorithm is stable in terms of bitrate switching and keeps re-buffering events at a minimum. This is expected, as all these QoE provisions have been taken into consideration with out modelling assumptions in Section 4. While further experimentation and comparison against similar low-latency bitrate adaptation techniques is required, these results are indicative of the powerful properties of *L2A-LL*, as a robust low-latency bitrate adaptation solution.

6 CONCLUSIONS

In this work we present *L2A-LL*, a novel bitrate adaptation algorithm based on online learning and the OCO theory, tailored for low-latency streaming. Overall, our proposed method performs well over a wide spectrum of network profiles in real experiments, due to its design principle; its ability to learn. It does so without requiring any parameter tuning, modifications according to application type or statistical assumptions for the channel; or throughput estimation. The *robustness* property of *L2A-LL* allows it to be classified in the small set of bitrate adaptation algorithms that mitigate the main limitation of existing HAS approaches; the dependence on statistical models for the unknowns. Real experiments show that *L2A-LL* reduces significantly latency, often to the near-second regime, while providing a high average streaming bit-rate without impairing the overall QoE. This result is independent of the channel and application scenarios. *L2A-LL* is lightweight and can be deployed even in mobile devices and allows for modular QoE prioritization, while concurrently it facilitates easy adjustments to consider other streaming application and/or user classes. This is of significant relevance in the field of modern HAS, where OTT video service providers are continuously expanding their services to include more diverse user classes, network scenarios and streaming applications.

REFERENCES

[1] [n.d.]. *dash.js.* https://github.com/Dash-Industry-Forum/dash.js

[2] MMSys 2020. 2020. *Grand Challenge on Adaptation Algorithms for Near-Second Latency - Test environment.* https://github.com/twitchtv/acm-mmsys-2020-grand-challenge

[3] Elena Veronica Belmega, Panayotis Mertikopoulos, Romain Negrel, and Luca Sanguinetti. 2018. Online convex optimization and no-regret learning: Algorithms, guarantees and applications. *arXiv e-prints* (2018).

[4] Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, and Roger Zimmermann. 2019. Bandwidth Prediction in Low-Latency Chunked Streaming. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video* (Amherst, Massachusetts) *(NOSSDAV '19)*. Association for Computing Machinery, New York, NY, USA, 7–13. https://doi.org/10.1145/3304112.3325611

[5] T. Chen, Q. Ling, and G. B. Giannakis. 2017. An Online Convex Optimization Approach to Proactive Network Resource Allocation. *IEEE Transactions on Signal Processing* 65, 24 (Dec 2017).

[6] Cisco Visual Networking Index. 2019. Forecast and Trends, 2017–2022. *White Paper* (Feb. 2019).

[7] DASH-IF. 2020. DASH-IF Change Request: Live Services DASH-IF Live services. *Agreed CR* (March 2020).

[8] ETSI. 2020. ETSI TS 103 285 V1.3.1 (2020-02) DVB-DASH). *International Standard TS 103 285* (Feb. 2020).

[9] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. of ACM Conf. on SIGCOMM.* http://doi.acm.org/10.1145/2619239.2626296

[10] ISO/IEC. 2014. Dynamic adaptive streaming over HTTP (DASH). *International Standard 23009-1:2014* (May 2014).

[11] ISO/IEC. 2018. Common Media Application Format CMAF). *International Standard 23000-19:2018* (Jan. 2018).

[12] T. Karagkioules, C. Concolato, D. Tsilimantos, and S. Valentin. 2017. A Comparative Case Study of HTTP Adaptive Streaming Algorithms in Mobile Networks. In *Proc. ACM NOSSDAV.* 1–6.

[13] M. Katsarakis, R. C. Teixeira, M. Papadopouli, and V. Christophides. 2016. Towards a Causal Analysis of Video QoE from Network and Application QoS. In *Proc. ACM Internet-QoE.* 31–36.

[14] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas of Communication* (April 2014).

[15] Nikolaos Liakopoulos, Apostolos Destounis, Georgios Paschos, Thrasyvoulos Spyropoulos, and Panayotis Mertikopoulos. 2019. Cautious Regret Minimization: Online Optimization with Long-Term Budget Constraints. In *Proc. of ICML* (Long Beach, CA, USA).

[16] Nikolaos Liakopoulos, Georgios Paschos, and Thrasyvoulos Spyropoulos. 2019. No Regret in Cloud Resources Reservation with Violation Guarantees. In *Proc. IEEE INFOCOM* (Paris, France).

[17] Michael J. Neely and Hao Yu. 2017. Online Convex Optimization with Time-Varying Constraints. *arXiv e-prints* (Feb. 2017).

[18] R. Pantos and W. May. 2018. RFC 8216 HTTP live streaming. *IETF, Request for Comments* (Aug. 2018).

[19] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. 2015. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys Tutorials* 17, 1 (Firstquarter 2015).

[20] Shai Shalev-Shwartz. 2012. Online Learning and Online Convex Optimization. *Foundations and Trends on Machine Learning* (Feb. 2012).

[21] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM.*

[22] T. Karagkioules et al. 2020. *Learn2Adapt-LowLatency.* Retrieved April 6, 2020 from https://github.com/unifiedstreaming/Learn2Adapt-LowLatency

[23] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proc. ACM Int. Conf. on SIGCOMM.* 14.

[24] C. Zhou, C. Lin, and Z. Guo. 2016. mDASH: A Markov Decision-Based Rate Adaptation Approach for Dynamic HTTP Streaming. *IEEE Transactions on Multimedia* (April 2016).

[25] Martin Zinkevich. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proc. of ICML.*