

Online detection of stalling and scrubbing in adaptive video streaming

Lorenzo Maggi, Jeremie Leguay, Michael Seufert, Pedro Casas

Angaben zur Veröffentlichung / Publication details:

Maggi, Lorenzo, Jeremie Leguay, Michael Seufert, and Pedro Casas. 2019. "Online detection of stalling and scrubbing in adaptive video streaming." In 2019 International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT), 3-7 June 2019, Avignon, France, 1-8. Piscataway, NJ: IEEE. <https://doi.org/10.23919/wiopt47501.2019.9144108>.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/>



Online Detection of Stalling and Scrubbing in Adaptive Video Streaming

Lorenzo Maggi^{*}, Jérémie Leguay[†], Michael Seufert[‡], Pedro Casas[‡]

^{*}Nokia Bell Labs, France, [†]Huawei Technologies, France Research Center

[‡]AIT Austrian Institute of Technology GmbH, Vienna, Austria

Abstract—Whether it is for network engineering or business intelligence insight purposes, it is crucial for an Internet Service Provider (ISP) to infer the Quality of Experience (QoE) perceived by the end user during a video streaming session. Specifically, it is important to detect video stalls as soon as they occur, to rapidly take counter-measures such as re-allocating resources more fairly among users. Video stalls fall into two different classes: (i) those caused by poor network conditions and (ii) those caused directly by the user when *scrubbing* or dragging the video playback forwards or backwards. However, only the former type of stalls degrade the QoE perceived by the end user. Therefore, in this paper we propose a technique to detect and classify stall events by observing the packets associated to a streaming session. We solve a least squares problem to minimize the distance between the estimated chunk’s bitrate and the potential bitrate sequence that a plausible playback buffer dynamics would produce. This amounts to finding the maximally likely state sequence for a properly defined Hidden Markov Model. We propose two polynomial dynamic programming algorithms, one of which running in online fashion, computing the exact solution in the ideal case of complete and exact measurement set. We claim that our method is also applicable in an encrypted scenario, since it is robust with respect to the estimation error of a number of parameters, as we show via simulations.

I. INTRODUCTION

Due to both high popularity and demanding network requirements of video streaming services, Internet Service Providers (ISPs) face the double challenge to deliver streaming traffic efficiently, simultaneously satisfying their customers expectations. To do so, ISPs rely on the concept of Quality of Experience [1], which allows to quantify the subjective experience of their customers from objective measurements. In video streaming, the most severe QoE degradation consists of playback interruptions - so called *stalling*, caused by re-buffering events [2], [3], [4], along with the waiting time until the playback resumes. While the advent of HTTP adaptive streaming has been able to mitigate these issues by adapting the video bitrate to the network conditions, stalling is still the most annoying and prevalent QoE degradation for the end user. Stalling is not only detrimental for the overall user experience, but it is also highly correlated to viewer engagement [5].

The ever growing competition forces ISPs to integrate QoE metrics into the core of their network management systems, from the network monitoring layer up to the traffic engineering

one. Thus, ISPs are highly interested in detecting the occurrence of stalling events caused by network problems as soon as they occur, to take appropriate countermeasures. However, the trend towards encrypted traffic has made the monitoring process more challenging and cumbersome for ISPs. In fact, in the encrypted scenario it is no longer possible to rely on Deep Packet Inspection (DPI) approaches to analyze the video data contained in each packet and, finally, to reconstruct the streaming process and the video buffer [6]. Yet, some machine learning approaches have recently been proposed to detect stalling from encrypted traffic, e.g., [7], [8], [9].

However, stalling might not only be caused by poor network conditions, but also by user behavior. Indeed, a playback interruption also happens when a user decides to jump (or *scrub*) the video playback cursor to a position which is outside the range of currently buffered playtime. Therefore, ISPs have to be careful not to confuse scrubbing (here called *type II stalling*), with (*type I*) stalling, which is due to poor network conditions. Scrubbing does not deteriorate QoE, as the user is aware of it, as opposed to type I stalling which is an indicator of unsatisfying network conditions, hence requiring ISPs to take immediate action.

In the adaptive video streaming scenario, we are the first to propose an online model-based method to detect video stalling events and distinguish between those caused by poor network conditions (type I) and by user scrubbing (type II). The main technical problem to overcome is that both types of stall generate deceptively similar observations at the packet stream level, hence they are indistinguishable via, say, a simple threshold policy on the chunk bitrate level.

Our first contribution is an online polynomial-time ($\mathcal{O}(\#\text{chunks}^2)$) algorithm that is able to detect and distinguish between type I and type II stalls. Our procedure can be seen as a Maximum Likelihood (ML) estimation of the state sequence of the Hidden Markov Model associated to the playback buffer state. In practice, we minimize via dynamic programming the sum of squares of the difference between the chunks’ bitrate that are observed (or estimated) and those being produced by a plausible playback buffer dynamics. Then, we also present a more complex, but still polynomial ($\mathcal{O}(\#\text{chunks}^4)$) algorithm that can leverage side-information on the maximum number of scrubbing events in a session to improve the detection and classification accuracy. Finally, we show via simulations that our algorithms are robust with respect to the estimation error of a number of factors, as the user application streaming policy,

^{*}Part of this work has been done while L. Maggi was at Huawei

the chunk download instants, and the chunk bitrate estimation. This suggests that our approach is also viable in the encrypted streaming scenario.

The remainder of the paper is structured as follows. Section II outlines related works, and Section III presents the scenario. Section IV describes the problem, which is solved in Sections V-VI with the help of dynamic programming. Section VII presents numerical evaluations, and Section VIII concludes the paper. Please refer to Table I for the list of notation symbols.

II. RELATED WORKS

Several papers focused on the estimation of video streaming stalling, as this is a key impacting factor for the QoE perceived by the end user, as described in [10]. Authors in [11], [6] proposed a system to extract downloaded playtime based on DPI of network packets. Thanks to it, the buffered playtime at the client’s side can be reconstructed, thus allowing for stalling event detection. Similar approaches were adopted by [12], [13]. The work [14] relies on the ratio of playback time and download time to estimate stalling, but requires the total size of the video, which might not always be available. Recently, the P.1203 model [15] has been standardized to predict the MOS (Mean Opinion Score) of HAS from stream inspection, thereby, also considering stalling. [16] focuses on the reconstruction of buffered playtime at the video player side, as previously done in [11]. It consists in a threshold-based policy to notice gaps in the downloading of chunks and assumes that the player has moved to a new playback position.

The requirements for the detection of user-actions in HTTP(S) adaptive streaming at network level are analyzed in [17]. The key challenge there is to recognize the result of user actions (e.g., play, pause, seeking) and distinguish them from streaming-related phenomena. In the literature, user actions have been considered only as features in QoE models. For instance, [18] found that video impairments can trigger user interactions, such as pausing the streaming and scrubbing. They proposed to integrate the user-viewing activities to estimate subjective parameters in a classical MOS log-logistic function. Moreover, [19] investigated a feature engineering method to correlate user experience with user-perceived quality and user actions. To the best of our knowledge, we are the first to propose a method to detect user scrubbing and distinguish them from standard stalling in online fashion.

To infer QoE, a number of practical challenges needs to be addressed as most video traffic is nowadays fully encrypted and user actions cannot be read from control traffic. Several parameters are to be inferred from packet traces with limited available information (e.g., transport protocol, arrivals times, payload size). In addition, as a single video might be split across multiple flows, the identification and tracking of video sessions is crucial. Several solutions based on machine learning and expert rules have been proposed to extract information from encrypted streams, e.g., [20], [21], [7], [8], [9].

Authors in [20] estimate YouTube QoE from features derived from packet sizes, inter-arrival times and network

Symbol	Description
t_i	[s] download time of the i -th chunk
p_i	[MB] payload size of the i -th chunk
b_i	[MB/s] encoding bitrate of the i -th chunk
λ_i	[s] playback time in the application’s buffer at time t_i
i	$= \{0, 1\}$ whether the video is in stall at time t_i
s_i	$= \{0, 1\}$ whether a scrubbing occurs during $(t_{i-1}; t_i]$
\mathbf{c}_i	[MB/s] series of channel throughput $\{c_k\}_{k < i}$
$f(\lambda_i, \mathbf{c}_i)$	$:= b_{i+1}$, HAS streaming policy
θ	[s] if the playback is in stall, it resumes when $\lambda > \theta$
Y_i	$:= (\lambda_i, \psi_i)$ buffer state at time t_i
I	total number of chunks
$\hat{\cdot}$	ISP’s estimation of metric \cdot
C	[MB/s] average network throughput
σ_{HAS}	std of HAS buffer threshold estimation error
σ_{DT}	std of chunk download time estimation error
p_{BR}	prob. that a chunk bitrate is mistaken with an adjacent one

Table I
LIST OF NOTATION SYMBOLS

throughput. A similar approach in [21] relies on real cellular network measurements to predict typical QoE indicators for streaming services. [7] leverages the work in [16] to estimate video QoE metrics. [8] predicts initial delay, stalling, and video quality from the network traffic in windows of 10s. The considered features are derived from IP or TCP/UDP headers only. [9] detects stalling of the streamed video within time slots of 1s by using a stream-like analysis approach based on packet-level statistics of the network traffic. The individual predictions of each time slot can be aggregated to also obtain accurate stalling statistics on a session level. Our solution leverages existing work only to infer parameters such as the video chunk bitrate, chunk arrival times and streaming policy parameters from encrypted packet traffic. It is not based on machine learning but rather follows a model-based approach, going beyond simplistic threshold based policies, which have no theoretical support and are sub-optimal.

III. SCENARIO

In the HTTP Adaptive Streaming (HAS) context, video files are encoded at multiple bitrate levels, segmented into consecutive fragments and stored in a web server. In our terminology, a *chunk* is a video fragment encoded at a specific bitrate. At the end user’s side, the video client adopts a streaming technique (see Section III-B), deciding when the next chunk should be fetched and at which bitrate. Clearly, higher bitrate translates into a better video quality for the user, but also into a longer download delay, since the chunk’s payload size is bigger. Downloaded but yet un-watched chunks are stored in the client’s application buffer, whose dynamics are described in Section III-A.

A. Client buffer dynamics

Let us now formalize the evolution of the client’s buffer, containing the chunks that have been downloaded from the web server but that have not been played yet. In this paper we sample our observations at times $\{t_i\}_i$, where t_i is the download time of the i -th chunk. We call λ_i the total playback time of the chunks present in the buffer at time t_i . The video

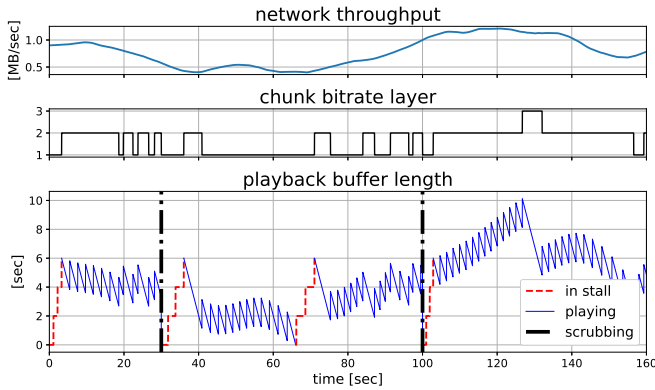


Figure 1. Buffer dynamics with scrubbing and stall events. We can already notice that a simple threshold policy on the chunk bitrate layer does not suffice to detect and classify stall events.

playback follows a hysteresis process: it *stalls* whenever the buffer empties, i.e., $\lambda = 0$, and it resumes when λ exceeds a threshold value θ , that is application-specific. We use the binary variable $\psi_i = \{0, 1\}$ to denote whether ($\psi_i = 1$) the playback is in stall mode or else ($\psi_i = 0$) the video is being regularly played at time t_i .

At any time, the user can drag the playback cursor ahead or backwards, to seek for a specific scene of interest. We call this user action *scrubbing*. We assume that whenever this happens, the video buffer empties completely, since the requested is not yet or no longer present in the application's buffer. We call $s_i = \{0, 1\}$ the binary variable that denotes whether ($s_i = 1$) at least a scrubbing occurs during interval $(t_{i-1}; t_i]$.

We describe the *state* of the buffer at the chunk download time t_i as the pair (λ_i, ψ_i) , denoting the playback length of the application's buffer and whether the playback is stalled at time t_i . At time 0, the buffer is empty and the playback is in stall, i.e., $(\lambda_0 = 0, \psi_0 = 1)$. Then, the buffer state evolves according to the recursive law:

$$(\lambda_i, \psi_i) = \Omega_i^{(s_i)} \left(\lambda_{i-1}, \psi_{i-1}, b_i, p_i, t_i - t_{i-1}, \theta \right) \quad (1)$$

where p_i and b_i are the payload size and the encoding bitrate of the i -th chunk, respectively, and the law $\Omega_i^{(s_i)}$ is defined as follows (see, e.g., [11] for a validation of the model):

$$\lambda_i = (1 - s_i) \left[\max \left(\lambda_{i-1} - (1 - \psi_{i-1})(t_i - t_{i-1}), 0 \right) + \frac{p_i}{b_i} \right] \quad (2)$$

$$\psi_i = \begin{cases} 1 & \text{if } (\psi_{i-1} = 1 \vee \lambda_i = \frac{p_i}{b_i}) \wedge \lambda_i < \theta \\ 0 & \text{else.} \end{cases}$$

In other words, upon the download of the i -th chunk, if the video is not in stall ($\psi_{i-1} = 0$) then the buffered playback time λ_i decreases by the time interval $(t_i - t_{i-1})$ elapsed since the last chunk download event. Also, λ_i increases by the playback time $\frac{p_i}{b_i}$ of the chunk just downloaded. Clearly, λ must be ensured to be non-negative; on the other hand, we remark that we do not need to upper bound λ (even if the buffer has finite length) since the process is driven by the successful download time of the chunks, that are hence ensured to fit in the buffer.

B. Client HAS streaming policy

The HAS application at the client's side implements a streaming policy that decides when the next chunk should be requested and at which encoding bitrate. We here assume for simplicity that a new chunk is requested as soon as the download of the previous chunk is terminated. Hence, we can describe the HAS policy as a function f that maps the current buffer length λ_i and the network throughput historic samples $\mathbf{c}_i = \{c_k\}_{k \leq i}$ into the bitrate b_{i+1} of the next chunk to be downloaded, i.e.,

$$b_{i+1} = f((1 - s_{i+1})\lambda_i, \mathbf{c}_i). \quad (3)$$

Typically, the channel history \mathbf{c}_i is used to predict the future channel evolution. The range of possible adaptive streaming policies is vast, but they can be categorized into three classes: *buffer-based*, *rate-based*, or *buffer-rate-based*, depending on whether $f(\cdot)$ is a function of the buffer occupation λ only, of the channel measurements \mathbf{c} only, or of both, respectively. In this paper we assume that the HAS policy is either *buffer-* or *buffer-rate-based*, since we need exploit the correlation between the chunks' bitrate and the buffer state.

For a pictorial intuition, in Fig. 1 we illustrate the buffer dynamics and the corresponding downloaded chunk bitrate layer in the presence of two scrubbing events.

C. Available measurements for an ISP in practice

In an operational scenario, the ISP only observes the packets associated to the streaming session between the video client and the web server. In the ideal case, the ISP would know when the session starts (at time 0, by convention) and, for each chunk $i = 1, \dots, I$, would measure its download time t_i , its payload size p_i , and obtain the corresponding bitrate b_i through content inspection. As such, the ISP could easily compute the playback duration of chunk i as the ratio p_i/b_i .

In practice, when the streaming session is encrypted, the ISP has first to detect and reconstruct the chunks from the sequence of monitored packets [7]. In addition, packets are normally intercepted on-the-fly within the network, hence potentially still far from the client destination [7]. These two facts imply that the *measured* chunk download times $\{\hat{t}_i\}$ do not coincide with the true ones $\{t_i\}_i$. Last but not least, chunks' bitrate $\{b_i\}_i$ is not directly accessible via packet inspection due to encryption, and it has to be estimated as $\{\hat{b}_i\}_i$, for example by modeling the specific streaming content [7] or by relying on statistical learning approaches [7].

Ideally, the ISP is fully aware of the client's HAS policy $f(\cdot)$ and the playback resume threshold θ . In practice, the ISP may be informed about the kind of streaming technique used by the client (e.g., whether it is buffer-, rate-, or buffer-rate based) but it can only estimate the actual shape of the HAS function $f(\cdot)$, denoted by $\hat{f}(\cdot)$, and the threshold $\hat{\theta}$.

IV. PROBLEM FORMULATION

In this paper we tackle the issue faced by an ISP aiming at detecting the stall events, possibly as soon as they occur,

and at classifying them into those of type I (caused by poor channel conditions) and of type II (due to user scrubbing).

We consider the practical scenario depicted in Section III-C, where due to non-ideal probing and encrypted traffic we can only estimate the parameters t, b, f, θ as $\hat{t}, \hat{b}, \hat{f}, \hat{\theta}$, respectively.

Our main idea is to infer the user's scrubbing behavior and the video stalls by computing the buffer length sequence and the scrubbing events that generate an array of chunks' bitrate being as "close" as possible to the estimated one $\{\hat{b}_i\}_i$ in terms of Euclidean distance. More formally, we aim to solve the following Least Squares (LS) problem:

$$\begin{aligned} \min_{\lambda, \psi, s} \sum_{i=1}^I \left[\hat{b}_i - \hat{f}((1-s_i)\lambda_{i-1}, \mathbf{c}_{i-1}) \right]^2 \quad (\text{LS}) \\ \text{s.t. } (\lambda_i, \psi_i) = \Omega_i^{(s_i)} \left(\lambda_{i-1}, \psi_{i-1}, \hat{b}_i, p_i, \hat{t}_i - \hat{t}_{i-1}, \hat{\theta} \right) \\ \lambda_0 = 0, \quad \psi_0 = 1, \end{aligned}$$

where the unknowns are λ , describing the buffer length dynamics, ψ and s , denoting the presence of a stall (type I or II) and of a scrubbing (type II stalls), respectively. The recursive buffer law Ω is defined as in (2). Once computed the optimal values of λ, ψ, s , the ISP can distinguish between type I and II stalls by simply comparing the scrubbing vector s and the instants i at which the buffer empties, i.e., $\lambda_i = 0$.

Hidden Markov Model (HMM) interpretation: In order to justify and provide further intuitions to our (LS) problem formulation for stall and scrubbing detection, we now give an interpretation that is founded on the theory of Hidden Markov Models. To this aim, let us first assume that the ISP can access the true measurements of t, b, f, θ . Under the hypothesis that the user scrubbing behavior described by $\{s_i\}_i$ is independent across time indexes i , the process evolving over the state pairs of the form (λ_i, ψ_i) is also Markovian. This fact stems directly from the buffer dynamics law defined in (LS). Also, if we define by π_{i+1} the probability that $s_{i+1} = 1$, then the state (λ_i, ψ_i, s_i) transitions to state $\Omega_{i+1}^{(s_{i+1})}(\lambda_i, \psi_i, \cdot)$ with probability π_{i+1} . In the HMM jargon, the tuples $\{\Omega_i^{(s_i)}\}_i$ define the *hidden* Markov model of the system at hand, depending on the unknown scrubbing vector s . On the other hand, one can observe the chunks' bitrate succession b which is a known function $f(\cdot)$ of the current state. We remark that the observed variable $\{b_i\}_i$ is *not* a Markov process *per se*.

Our task is to reconstruct the most likely succession of hidden states given the observed bitrate sequence. This can be equivalently formulated (see [22]) as the minimization of the Euclidean distance between the observations and the one produced by a guess of the hidden state succession, plus a regularization term depending on the scrubbing probability π :

$$\min_{\lambda, \psi, s} \sum_{i=1}^I \left[b_i - f((1-s_i)\lambda_{i-1}, \mathbf{c}_{i-1}) \right]^2 + \mu \log \left(\frac{1-\pi_i}{\pi_i} \right).$$

Since we do not have any *a priori* knowledge on the presence of scrubbing at a certain time instant, then we let $\pi_i = 0.5$, which brings us to our original formulation in (LS). The only

difference in our case resides in the fact that in (LS) we replace the true values t, b, f, θ with the estimated ones $\hat{t}, \hat{b}, \hat{f}, \hat{\theta}$.

V. SOLVING (LS) VIA DYNAMIC PROGRAMMING

In this section we present a dynamic programming approach to solve our main least squares problem (LS) in *polynomial* time and, remarkably, in an *online* fashion, as new chunks are observed. Following our HMM interpretation of the problem (LS) given in Section IV, we let the buffer length / stall pair $(\lambda_i, \psi_i) := Y_i$ play the role of the *state* of the system at time t_i . In a control-theoretic jargon, we think of the scrubbing vector s as the *control variable*, that drives the state of the system as close as possible to the observed one.

In order to explain our dynamic programming approach, let us now suppose that the system is in state $Y_{i-1} = (\lambda_{i-1}, \psi_{i-1})$ at time t_{i-1} . If the control variable $s_i = 1$, then the buffer empties and the playback stalls, hence at the next step t_i we have $Y_i = (0, 1)$. Otherwise, if $s_i = 0$ then the buffer state evolves by the law $Y_i = \Omega^{(0)}(Y_{i-1}, b_i)$. Thus, we can attach an additive cost $\delta^{(s_i)}$ to the transition $Y_{i-1} \xrightarrow{s_i} Y_i$ controlled by the scrubbing variable s_i , and defined as the squared difference between the current estimated chunk bitrate and the one produced by the current plausible state:

$$\begin{aligned} Y_{i-1} \xrightarrow{s_i} Y_i = \Omega^{(s_i)}(Y_{i-1}, \dots), \text{ with additive cost:} \\ \delta^{(s_i)}(Y_{i-1}, Y_i) = \left| \hat{b}_i - f((1-s_i)\lambda_{i-1}, \mathbf{c}_{i-1}) \right|^2. \quad (4) \end{aligned}$$

Our problem then becomes a min-cost path one, that requires to compute the discrete (buffer) state and (scrubbing) control succession with minimal cost. To this aim, we resort to the celebrated Bellman's principle [23]. Let us first define $\delta^*(Y_i)$ as the minimum total cost to reach state Y_i from the initial state $Y_0 = (0, 1)$. Naturally, we let $\delta^*(Y_0) := 0$. Then, the Bellman equation defines $\delta^*(Y_i)$ by induction as the minimum cost to reach Y_i from any potential previous state Y_{i-1} and for both choices of the variable $s_i = \{0, 1\}$:

$$\delta^*(Y_i) = \min_{\substack{Y_{i-1} \in X_{i-1}, s_i \in \{0, 1\}: \\ Y_i = \Omega_i^{(s_i)}(Y_{i-1})}} \delta^*(Y_{i-1}) + \delta^{(s_i)}(Y_{i-1}, Y_i), \quad (5)$$

where X_i is the set of *reachable* states at time step t_i for a certain choice of the control variables s_1, \dots, s_i .

In order to actually solve the Bellman equation (5) we use a forward induction technique and we find the minimum cost $\delta^*(Y_i)$ for successive values of the chunk index $i = 1, \dots, I$. Then, we compute the optimal final state Y_I^* as:

$$Y_I^* = \arg \min_{Y_I \in X_I} \delta^*(Y_I) \quad (6)$$

and we retrieve the optimal buffer state and scrubbing vector in backward fashion starting from Y_I^* . We provide the details of the forward-induction procedure in Algorithm 1.

Complexity: Classically, dynamic programming techniques suffer from the curse of dimensionality [23] which denotes the exponential growth of the state space as size of the input (in our case, the number of time intervals I) increases. On the one hand, in our model there exist 2^I possible paths from the initial

Algorithm 1: Least Squares (LS) Algorithm

- 1 Set $Y_0 = (\lambda_0, \psi_0) := (0, 1)$, $\delta^*(Y_0) := 0$, $X_0 := \{Y_0\}$;
 - for** $i = 1, \dots, I$ **do**
 - 2 Generate all reachable states X_i at time t_i :
 $X_i = \{Y_i = \Omega_i^{(s_i)}(Y_{i-1}), \forall Y_{i-1} \in X_{i-1}, s_i = 0, 1\}$.
 - 3 Compute the minimum cost $\delta^*(Y_i)$ by solving the Bellman equation (5), for all $Y_i \in X_i$.
 - 4 Compute the predecessor $(Y_{i-1}, s_i) = \text{pred}(Y_i)$ as the arg min of Equation (5), for all $Y_i \in X_i$.
 - 5 Compute the optimal final state Y_I^* as in (6).
 - 6 Retrieve the optimal buffer dynamics and scrubbing times in backward fashion:
 - 7 **for** $j = 1, \dots, I$ **do**
 - 8 Compute $(Y_{I-j}^*, s_{I-j}^*) = \text{pred}(Y_{I-j+1}^*)$
 - 9 **return** the Least Squares scrubbing vector s_i^* and buffer state $Y_i^* := (\lambda_i^*, \psi_i^*)$ for all $i = 1, \dots, I$
-

state Y_0 to the final stage $i = I$ on the state transition graph defined by Equation (4), which correspond to all the binary scrubbing vector combinations. On the other hand, Bellman's equation (5) allows us to prune at least half of the possible paths generated at each step i . This implies that the maximum number of feasible states $|X_i|$ is at most $i + 1$ and that the overall complexity is quadratic in the number of steps I .

Lemma 1. *Algorithm 1 solves exactly the Least Squares problem (LS) with complexity $\mathcal{O}(I^2)$.*

Proof. The optimality of Algorithm 1 has already been shown via the Bellman principle in (5). To prove its complexity, we first observe that, according to the buffer dynamics rule $\Omega^{(s_i)}$ described in (2), the control $s_i = 1$ forces all states of the kind $Y_{i-1} \in X_{i-1}$ to converge to the empty buffer state $(\lambda_i = 0, \psi_i = 1) = \Omega_i^{(1)}(Y_{i-1}, b_i)$. Then, such state has at least $|X_{i-1}|$ potential predecessors, and only one is chosen via Bellman equation (5) with complexity linear in $|X_{i-1}|$. All other states $Y_i \in X_i$ have only one possible predecessor. It stems that i) the total complexity of Algorithm 1 at step i is linear in $|X_{i-1}|$. Next, we notice the set of feasible states X_i is a function of the previous states X_{i-1} as:

$$X_i = \{Y_i = \Omega^{(s_i=0)}(Y_{i-1}, b_i), \forall Y_{i-1} \in X_{i-1}\} \cup \{(0, 1)\}.$$

Therefore, $|X_i| \leq |X_{i-1}| + 1$ and thus $ii) |X_i| \leq i + 1$. Since there are I steps, the thesis directly follows from $i), ii)$. \square

Online detection: It is crucial to observe that the LS detection can be performed in online fashion, to detect scrubbing and stall events as soon as they occur, by slightly modifying Algorithm 1. In fact, the induction procedure in steps 1-4 of Algorithm 1 only requires a *single* forward pass on the input data (i.e., the bitrate \bar{b} , the download times t , the chunk payload p and channel throughput c). Then, if one wishes to solve (LS) at each intermediary step t_i , it suffices to retrieve at each time t_i in backward fashion the LS buffer dynamics

and scrubbing vector via the steps 6-8 of Algorithm 1. So, the only variation consists in plugging lines 5-8 into the main **for** loops. Since the complexity of backward retrieval is linear in the number of chunks i observed up to time t_i , and since this is carried out I times, it stems that the overall complexity of the online version of Algorithm 1 is still $\mathcal{O}(I^2)$.

Corollary 1. *The online version of Algorithm 1, computing the intermediary optimal solution from step 1 to each intermediary step $i = 1, \dots, I$, also has overall complexity $\mathcal{O}(I^2)$.*

VI. CONSTRAINING THE NUMBER OF SCRUBBINGS

In this section we make the further assumption that the ISP is able to access some side information on the user's scrubbing behavior, and more specifically on the total number of scrubbing events $\sum_{i=1}^I s_i$ being at most \bar{S} . Such information may be extracted by analyzing the statistical properties of observed features (i.e., bitrate and channel throughput) in an *a posteriori* fashion, after the whole streaming session dataset has been observed, as advocated in [17]. For this reason, we expect that the technique proposed in this section is not suitable for online stall detection purposes. We wish to exploit such side information to improve the accuracy of our approach and, more specifically, reduce the occurrence of false positive scrubbing detection of our approach.

Our next goal is then to solve the following Constrained Least Squares (CLS) problem:

$$\text{Solve (LS) s.t. } \sum_{i=1}^I s_i \leq \bar{S}. \quad (\text{CLS})$$

Using the dynamic programming jargon and the notation introduced in the Section V, (CLS) requires to solve a min-cost path from the initial state Y_0 to the final step $i = I$, under the the hard constraint that the control variable s_i can be set to 1 at most \bar{S} times. The constrained min-cost path problem is known to be NP-hard [24] for general graphs; in our special case, though, it can be solved in polynomial time by carefully adapting Algorithm 1 to the constrained case.

Solving (CLS) via Dynamic Programming: We now adapt the dynamic programming Algorithm 1 used to solve (LS) to solve the constrained problem (CLS) in polynomial time. For this purpose, we embed the number of scrubbing events $S_i = \sum_{k=1}^i s_k$ accumulated up to the current step t_i in the definition of *state* Y_i' of the system, which then becomes the triple $Y_i' = (\lambda_i, \psi_i, S_i)$. The transition law between states now writes:

$$\begin{aligned} Y_{i-1}' &= (\lambda_{i-1}, \psi_{i-1}, S_{i-1}) \xrightarrow{s_i} Y_i = (\lambda_i, \psi_i, S_i) \quad (7) \\ \text{s.t. } (\lambda_i, \psi_i) &= \Omega^{(s_i)}(\lambda_{i-1}, \psi_{i-1}) \\ S_i &= S_{i-1} + s_i. \end{aligned}$$

If $S_i > \bar{S}$ then the transition does not exist since it violates the constraint. The cost $\delta^{(s_i)}(Y_{i-1}', Y_i')$ associated to the state transition is defined exactly as in (4), as the number of

accumulated scrubblings does not affect the objective function. The Bellman's equation underlying (CLS) then becomes:

$$\delta^*(Y'_i) = \min_{\substack{Y'_i \in X_{i-1}, s_i \in \{0,1\}: \\ (\lambda_i, \psi_i) = \Omega_i^{(s_i)}(\lambda_{i-1}, \psi_{i-1}, \dots) \\ S_i = S_{i-1} + s_i \leq \bar{S}}} \delta^*(Y'_{i-1}) + \delta^{(s_i)}(Y'_{i-1}, Y'_i) \quad (8)$$

where the initial state is now $Y'_0 = (\lambda_0 = 0, \psi_0 = 1, S_i = 0)$ with associated optimal cost $\delta^* = 0$. Bellman's equation (8) for (CLS) can be solved by using the forward induction procedure, similarly to what has been shown in Algorithm 1.

Algorithm 2: Constrained Least Squares (CLS) Algorithm

- 1 Analogous to Algorithm 1 with two variants: the new state transition is defined as (7) and the new Bellman's equation is (8)
-

Due to the enlarged state space, the complexity of Algorithm 2 is higher than for Algorithm 1, but it is still polynomial in the input size, and more specifically quartic in the number I of downloaded chunks.

Lemma 2. *Algorithm 2 solves exactly the constrained Least Squares problem (CLS) with complexity $\mathcal{O}(I^4)$.*

Proof. Let X'_i be the set of reachable states at time step t_i . Let us split X'_i into two categories, $X_i^{A'}$ and $X_i^{B'}$. We denote by $X_i^{B'}$ the states having an empty buffer, i.e., $X_i^{B'} = X_i \cap \{(0, 1, S_i)\}_{S_i=1, \dots, \bar{S}}$. Instead, $X_i^{A'} = X_i \setminus X_i^{B'}$ is the set of states generated by the absence of scrubbing during the last interval ($s_i = 0$). In order to bound the number of states at step i , we then notice that the next two relations holds:

$$|X_i^{A'}| \leq |X_{i-1}^{A'}| + |X_{i-1}^{B'}| \quad (9)$$

$$|X_i^{B'}| \leq \bar{S}, \quad \forall i = 1, \dots, I, \quad (10)$$

where (9) follows from the fact that the set of possible predecessors of $X_i^{B'}$ is a subset of $X_{i-1}^{B'} \cup X_{i-1}^{A'}$. Since at the boundary $i = 0$ we have $|X_0^{A'}| = 0$ and $|X_0^{B'}| = 1$, then we deduce from (10-9) that

$$|X_i^{A'}| \leq i\bar{S}. \quad (11)$$

Next, we upper bound the complexity $C(i)$ of iteration i of Algorithm 2. Since all states in $X_i^{A'}$ have only one predecessor, they just need to be generated, in a time linear in $X_i^{A'}$. On the other hand, to find the optimal predecessor for each of the state $Y'_i \in X_i^{B'}$ one needs to solve the Bellman equation (8), with linear complexity in the number of possible predecessors, being at most $|X_{i-1}^{A'}| + 1$. Therefore,

$$\begin{aligned} C(i) &\leq |X_i^{A'}| + (|X_{i-1}^{A'}| + 1)|X_i^{B'}| \\ &\leq i(\bar{S}^2 + \bar{S}) + \bar{S} - \bar{S}^2, \end{aligned} \quad (12)$$

where (12) stems from (11) and (10). Finally, we can bound the total complexity of the algorithm as $\sum_{i=1}^I C(i) \in \mathcal{O}((\bar{S}^2 + \bar{S})I^2)$. Finally, since $\bar{S} \leq I$ we conclude that the overall complexity is $\mathcal{O}(I^4)$, which proves the thesis. \square

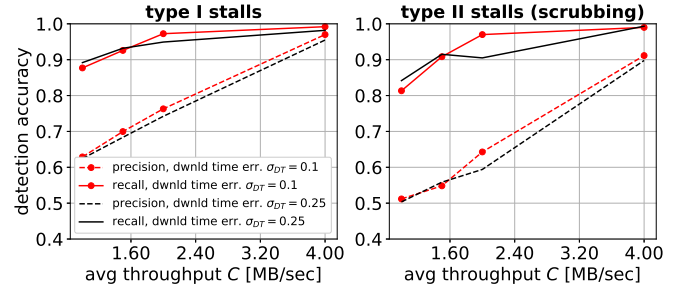


Figure 2. LS Algorithm 1 robustness against estimation error on download times. HAS policy threshold estimation error std $\sigma_{HAS} = 0.1$, probability of chunk bitrate estimation error $p_{BR} = 0.1$.

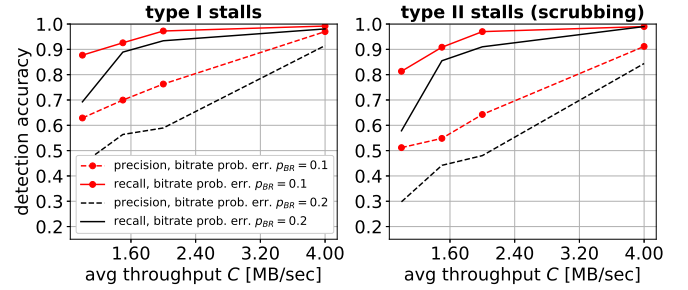


Figure 3. LS Algorithm 1 robustness against estimation error on chunk bitrate. HAS policy threshold estimation error std $\sigma_{HAS} = 0.1$, chunk download time error std $\sigma_{DT} = 0.1$.

VII. NUMERICAL EVALUATIONS

After presenting our stalling detection and classification algorithms and their complexity, we now turn to evaluating their performance via extensive numerical simulations. In particular, we focus on showing the robustness of our approaches with respect to the estimation error of three key factors, i.e., the chunk download time, the shape of HAS policy and the chunks' encoding bitrate.

Simulation settings: In order to test the performance of our algorithms, we generated synthetic data on a realistic HAS scenario that we now describe. We consider a threshold buffer-based HAS policy with playback resume threshold $\theta = 5s$ and with three available chunk encoding bitrate levels. More specifically, when the buffer contains a playback time $\leq 10s$ (within (10s, 20s) and $\geq 20s$, respectively) the HAS application fetches a chunk with encoding bitrate equal to 0.5MB/s (1MB/s and 3MB/s, respectively). The application buffer capacity is equivalent to 30 playback seconds. The network throughput c_i is generated via an ARIMA model with average value C MB/s. In each simulated streaming session the user scrubs the video 3 times at uniformly random times over a whole time horizon of 5 minutes. All measurement points are averages over 100 trials.

Robustness w.r.t. estimation errors: In Fig. 2-6 we evaluate the robustness of LS Algorithm 1 with respect to the estimation errors on chunk bitrate, HAS policy and chunk download time, for different values of the average network throughput. We

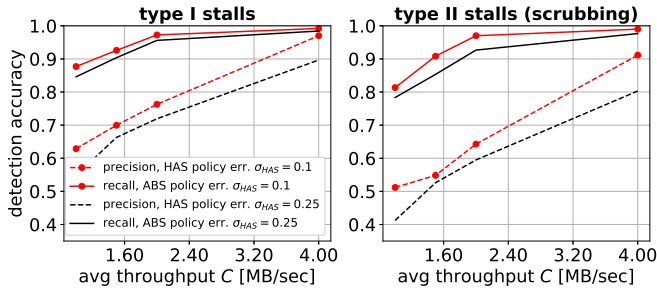


Figure 4. LS Algorithm 1 robustness against estimation error on HAS policy. Chunk download time error std $\sigma_{DT} = 0.1$, probability of chunk bitrate estimation error $p_{BR} = 0.1$.

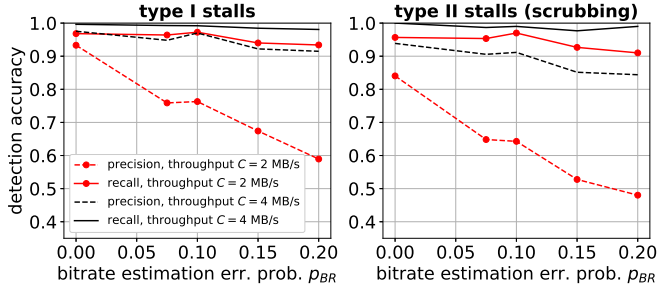


Figure 5. LS Algorithm 1 robustness against estimation error on chunk bitrate. Chunk download time error std $\sigma_{DT} = 0.1$, HAS policy threshold estimation error std $\sigma_{HAS} = 0.1$.

measure the accuracy of type I and II stall detection in terms of precision and recall¹. In our simulations, the stall detection is deemed correct if it falls at most 2 chunk download intervals away from the correct one.

Let us now describe how we emulate estimation errors. The chunk bitrate level \hat{b} is mistaken with an adjacent level (i.e., layer 1 \rightarrow 2, layer 2 \rightarrow 1,3, layer 3 \rightarrow 1) with a certain probability p_{BR} , independently of each other. The estimation error on HAS thresholds of the streaming policy is Gaussian with standard deviation equal to σ_{HAS} times the difference between two consecutive thresholds (in our case, 10s). Similarly, the estimated chunk download times are affected by an additive Gaussian *i.i.d.* error with standard deviation equal to σ_{DT} times the interval between consecutive download instants. This models the fact that packets are intercepted on-the-fly by a probe within the network, hence not at the application side. As shown in Fig. 2-6, the LS algorithm shows remarkable robustness with respect to chunk download time and HAS policy. Our method is more sensitive to the errors on chunk bitrate (see Fig. 5), which deteriorates the precision of stall and scrubbing detection. Such behavior is expected, since our least squares approach relies on chunk bitrate estimation to imitate the observed chunk bitrate succession with a plausible one, to finally infer the buffer evolution.

We also observe that the detection accuracy improves as the channel throughput C increases. Intuitively, with high values

¹precision = $\frac{\text{true positives}}{\text{true} + \text{false positives}}$, recall = $\frac{\text{true positives}}{\text{true pos.} + \text{false neg.}}$

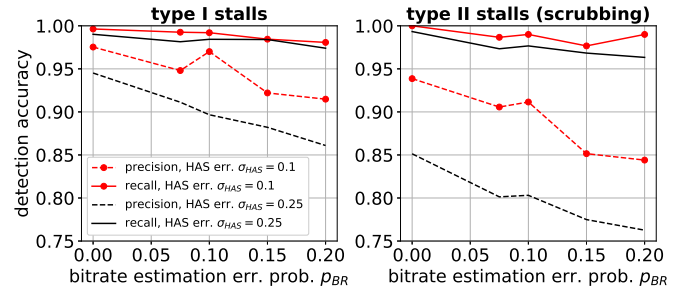


Figure 6. LS Algorithm 1 robustness against estimation error on chunk bitrate and HAS policy. Chunk download time error std $\sigma_{DT} = 0.1$, average network throughput $C = 4$ MB/s.

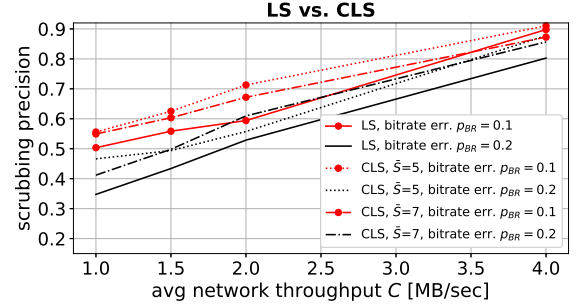


Figure 7. LS Algorithm 1 vs. CLS Algorithm 2 scrubbing detection performance. HAS policy threshold estimation error std $\sigma_{HAS} = 0.1$, chunk download time error std $\sigma_{DT} = 0.1$. Notice that CLS allows to limit the false positive occurrences.

of C , the buffer fill up more quickly, which translates in higher chunk bitrate for any buffer- or buffer-rate based policy. In this case, stalls, which generally produce low bitrate observations, are thus more easily distinguishable for our method.

Increasing scrubbing precision via (CLS): As observed earlier on, although robust to estimation errors, the LS algorithm has tendency to overestimate the number of stall events in the presence of important errors on chunk bitrate estimation. This hence generates too many false positives which reflects into lower precision. By upper limiting the possible number of scrubblings \bar{S} ($= 5, 7$) via CLS Algorithm 2, the scrubbing detection precision improves, as we can appreciate in Fig. 7. However, this comes with an increase of complexity, from $\mathcal{O}(I^2)$ for (LS) to $\mathcal{O}(I^4)$ for (CLS), as shown in Lemma 2.

Online detection: We demonstrate the capability of Algorithm 1 to solve the (LS) detection problem in online fashion, as outlined in Section V. As new chunks' bitrate are observed, LS Algorithm 1 is able to produce new estimates of the stall and scrubbing vectors, without the need of solving the whole optimization problem (LS) from scratch. In Fig. 8 we show the detection accuracy of ML algorithm as time goes by, with three scrubbing events at time 50s, 130s and 210s. Scrubbing can be indeed detected on-the-fly, with a small delay of a couple of seconds that is visible through the temporary drop in the scrubbing detection precision. The detection delay is due to a common feature of dynamic control problems: as the decision instant approaches the end of the time window, the

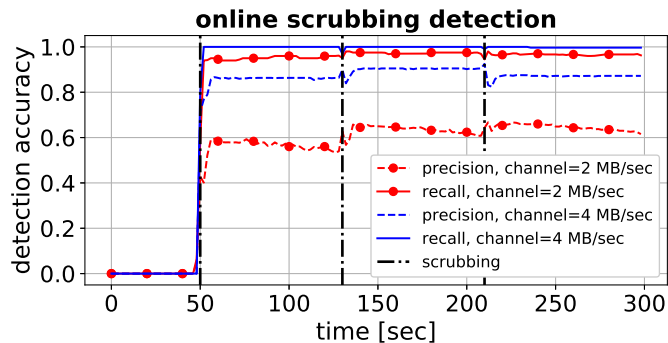


Figure 8. **Online scrubbing detection** of LS Algorithm 1, with chunk estimation probability error $p_{BR} = 0.2$ and three scrubblings (at the vertical black dashed lines)

dynamic programming decision become more greedy, hence more prone to errors.

VIII. CONCLUSIONS

Detecting video stalling is of paramount importance to the ISP, for network engineering or business intelligence insight purposes, and in particular to rapidly detected end-user QoE degradation. However, one has to distinguish between the stalls caused by poor network conditions and those due to user-initiated actions such as video-player scrubbing, since only the former one contributes to degrade the user's QoE. Motivated by this, we have proposed two polynomial algorithms that, by reconstructing the most plausible buffer evolution, detect the stall events and decide whether each stall is caused by user scrubbing or not. Both algorithms solve a least squares problem that finds its roots in the Hidden Markov Model theory. The former approach (LS, Alg. 1) can be executed in online fashion, to detect stall and scrubbing events as soon as they occur. The latter (CLS, Alg. 2) can exploit side information from the user's behavior on the maximum number of scrubbing events within a session. Our simulations suggest that the proposed methods are practical and suitable for the encrypted setting, since they are sufficiently robust with respect to estimation errors on a number of parameters, as the chunks' bitrate, the chunk download times and the client's streaming policy. As a future plan, we plan to plug our method in a real HAS system with encrypted traffic, where chunk bitrate is estimated via supervised learning approaches.

REFERENCES

- [1] P. Le Callet, S. Möller, and A. Perkis (eds), "Qualinet White Paper on Definitions of Quality of Experience," European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003), Tech. Rep., 2013, version 1.2.
- [2] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Höbfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.
- [3] D. Ghadiyaram, J. Pan, and A. C. Bovik, "A Time-varying Subjective Quality Model for Mobile Streaming Videos with Stalling Events," in *Proceedings of SPIE Applications of Digital Image Processing XXXVIII*, 2015.

- [4] K. Zeng, H. Yeganeh, and Z. Wang, "Quality-of-experience of Streaming Video: Interactions between Presentation Quality and Playback Stalling," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2016.
- [5] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *Proceedings of the ACM SIGCOMM*, 2011.
- [6] P. Casas, M. Seufert, and R. Schatz, "YOUQMON: A System for Online Monitoring of YouTube QoE in Operational 3G Networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.
- [7] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura, "eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic," in *Network Traffic Measurement and Analysis Conference (TMA)*, 2018.
- [8] M. H. Mazhar and M. Z. Shafiq, "Real-time video quality of experience monitoring for https and quic," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.
- [9] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Stream-based Machine Learning for Real-time QoE Analysis of Encrypted Video Streaming Traffic," in *3rd International Workshop on Quality of Experience Management*, 2019.
- [10] T. Höbfeld, R. Schatz, M. Seufert, M. Hirth, T. Zinner, and P. Tran-Gia, "Quantification of YouTube QoE via Crowdsourcing," in *Proceedings of the International Workshop on Multimedia Quality of Experience - Modeling, Evaluation, and Directions (MQoE)*, 2011.
- [11] R. Schatz, T. Höbfeld, and P. Casas, "Passive YouTube QoE Monitoring for ISPs," in *IMIS*, 2012, pp. 358–364.
- [12] P. Azeiteiras, A. Azcona-Rivas, J. Navarro-Ortiz, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A Simple Model for Predicting the Number and Duration of Rebuffering Events for YouTube Flows," *IEEE Communications Letters*, vol. 16, no. 2, pp. 278–280, 2012.
- [13] M. Eckert, T. M. Knoll, and F. Schlegel, "Advanced MOS Calculation for Network Based QoE Estimation of TCP Streamed Video Services," in *Proceedings of the 7th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2013.
- [14] P. Szilágyi and C. Vulkán, "Network side Lightweight and Scalable YouTube QoE Estimation," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2015.
- [15] International Telecommunication Union, "ITU-T Recommendation P.1203: Parametric Bitstream-based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services over Reliable Transport," 2016. [Online]. Available: {<https://www.itu.int/rec/T-REC-P.1203/en>}
- [16] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, "BUFFEST: Predicting buffer conditions and real-time requirements of HTTP (S) adaptive streaming clients," in *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 2017, pp. 76–87.
- [17] Sandvine, "Industry White Paper: Video Quality of Experience: Requirements and Considerations for Meaningful Insight," Tech. Rep., 2016. [Online]. Available: <https://www.sandvine.com/hubfs/downloads/archive/whitepaper-video-quality-of-experience.pdf>
- [18] R. K. Mok, E. W. Chan, X. Luo, and R. K. Chang, "Inferring the QoE of HTTP video streaming from user-viewing activities," in *Proceedings of the first ACM SIGCOMM workshop on Measurements up the stack*. ACM, 2011, pp. 31–36.
- [19] T. Yue, H. Wang, and S. Cheng, "Learning from users: a data-driven method of QoE evaluation for Internet video," *Multimedia Tools and Applications*, pp. 1–32, 2018.
- [20] I. Orsolich, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A Machine Learning Approach to Classifying YouTube QoE based on Encrypted Network Traffic," *Multimedia Tools and Applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.
- [21] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring video QoE from encrypted traffic," in *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016, pp. 513–526.
- [22] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.
- [23] R. E. Bellman, "Dynamic Programming," *Princeton University Press*, 1957.
- [24] M. R. Garey and D. S. Johnson, *Computers and intractability*. Freeman New York, 2002, vol. 29.