This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2019.2914388, IEEE Transactions on Circuits and Systems for Video Technology

FastScan: Robust Low-Complexity Rate Adaptation Algorithm for Video Streaming over HTTP

Anis Elgabli and Vaneet Aggarwal

***Abstract*—This paper proposes and evaluates a novel algorithm for streaming video over HTTP. The problem is formulated as a non-convex optimization problem which is constrained by the predicted available bandwidth, chunk deadlines, available video rates, and buffer occupancy. The objective is to optimize a QoE metric that maintains a tradeoff between maximizing the playback rate of every chunk and ensuring fairness among different chunks for the minimum re-buffering time. We propose FastScan, a low complexity algorithm that solves the problem. Online adaptations for dynamic bandwidth environments are proposed with imperfect available bandwidth prediction. Results of experiments driven by Variable Bit Rate (VBR) encoded video, video platform system (dash.js), and cellular bandwidth traces of a public dataset reveal the robustness of the online version of FastScan algorithm and demonstrate its significant performance improvement as compared to the considered state-of-the-art video streaming algorithms. For example, on an experiment conducted over 100 real cellular available bandwidth traces of a public dataset that spans different available bandwidth regimes, our proposed algorithm (FastScan) achieves the minimum re-buffering (stall) time and the maximum average playback rate in every single trace as compared to Bola, Festive, BBA, RB, and FastMPC, and Pensieve algorithms.**

***Index Terms*—Video Streaming, Non-Convex Optimization, Adaptive Bit Rate (ABR), Average Video Quality, Stall Duration**

I. Introduction

The mobile video traffic is estimated to increase by 9x between 2016 and 2021 [1]. A convergence of technological, business and social factors are contributing to this trend. These include the ubiquity of smartphones and tablets, high-speed cellular connectivity, the increasing availability of “over the top” video content, and a marked shift in user consumption preferences. Despite numerous adaptive streaming algorithms being devised and deployed, the video quality under mobility is in many cases unacceptably poor [2]. This paper gives a novel algorithm for adaptive video streaming that aims to improve the quality of experience for the end-user.

In the past decade, a lot of work from both research and industry has focused on the development of *adaptive* video encoding in which the video content on the server side is divided into chunks. Each chunk is then encoded into multiple quality levels and *adaptive* video streaming techniques can dynamically adjust the quality of the video being streamed to the changes in network conditions. The rate adaptive schemes dynamically switch between the different available quality levels based on the network condition, the client buffer occupancy, *etc.*

The recent adoption of the open standards MPEG-DASH [3] has made Adaptive bit-rate (ABR) video streaming the most popular video streaming solution. Commercial systems such as Apples’ HLS [4], Microsoft’s Smooth Streaming [5], and Adobe’s HDS [6] are all some variants of ABR streaming techniques. In recent studies, researchers have investigated various approaches for making streaming decisions, for example, by using control theory [7], [8], Markov Decision Process [9], machine learning [10], client buffer information [11], and data-driven techniques [12], [13]. In this paper, we use the prediction of future network condition to provide a novel algorithm for ABR video streaming.

In [7], a similar optimization problem is considered. However, the proposed optimization formulation in [7] is not shown to be optimally solvable in polynomial complexity. Moreover, a lookup table based on approximating the solution of the offline problem for a given set of encoding rates is proposed. To reduce the table size, the authors suggest dividing the offline solution to bins, so that the final decision is an approximation to the offline solution. However, the table is only valid for a specific set of encoding rates, hence another table need to be generated and stored for a different set of rates. Recently, the authors of [14] proposed a system that generates ABR algorithms using reinforcement learning (RL). Their idea is to train a neural network (NN) model that selects bitrates for future video chunks based on observations collected by client’s video players. The RL based system uses the metric described in [7] as the QoE metric that the RL agent needs to learn how to optimize. However, such a system needs a lot of training data including videos encoded at different rates and bandwidth traces that span wide range of network bandwidth conditions.

In this paper, we propose a QoE metric that accounts for the diminishing QoE gain as we go higher in the playback rate (quality level), *i.e.,* we consider the concave nature of the user’s QoE with respect to the playback rate. We formulate the quality decisions of the video chunks as an optimization problem. Moreover, we develop a low complexity algorithm to solve the proposed problem, and we show that the algorithm solves the optimization problem optimally when every quality level is constant Bit Rate (CBR) encoded (*i.e.,* when the n-th quality level rate is same for all chunks). We further show

A. Elgabli is with the School of Electrical Computer Engineering, Purdue University, West Lafayette IN, USA, email: aelgabli@purdue.edu. V. Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette IN, USA, email: vaneet@purdue.edu. This work was supported in part by the U.S. National Science Foundation under grants CCF-1527486 and CNS-1618335.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2019.2914388, IEEE Transactions on Circuits and Systems for Video Technology

that the algorithm significantly outperform the-state-of-the-art algorithms ABR streaming algorithms. The formulation and the algorithm we propose in this paper are modifications of the formulation and algorithm for SVC (Scalable Video Coding) encoded videos in [15]. In this work, we consider AVC encoded videos in which every chunk is encoded into independent versions that represent different qualities. Moreover, in contrast to the work in [15], we implement the proposed algorithm in this paper in a real system with real videos that are AVC encoded.

Since the available bandwidth can only be predicted for short time ahead with prediction error, the proposed algorithm uses the short prediction to make quality decisions for W chunks ahead, and repeats after the download of every chunk to adjust for prediction errors and to include one more chunk ahead every time. Thus, the algorithm is a sliding window based algorithm. The complexity of the algorithm is linear in W, and in contrast to [7], the approach does not require to pre-store information about different encoding rates. The main contributions are summarized as follows.

• We formulate the video streaming over HTTP constrained by the predicted available bandwidth, chunk deadlines, available video rates, and buffer occupancy as a non-convex optimization problem. The objective is to optimize a QoE metric that maintains a tradeoff between maximizing the playback rate of every chunk and ensuring fairness among all chunks for the minimum stall duration.

• We develop FastScan, a novel low-complexity algorithm to solve the video streaming problem. FastScan has a complexity that is linear in the prediction window size. The algorithm is optimal if the video is CBR encoded. The online adaptation of the algorithm is re-run after the download of every chunk to re-consider and decide the quality of the next window of chunks.

• Real implementation test bed with the open source video framework dash.js shows significant performance improvement of our algorithm as compared to the-state-of-art algorithms. For example, on an experiment conducted over three sets of real cellular bandwidth traces of public datasets that spans different available bandwidth regimes, our proposed algorithm (FastScan) achieves the highest QoE in more than 99% of traces in almost every dataset as compared to Festive [16], BBA [11], RB [7], and Bola [17], FastMPC [7], and Pensieve [14].

The rest of the paper is organized as follows. Section II formulates the problem, and describes the notations. Section III describes the proposed algorithm. Section IV describes the implementation and shows that the proposed algorithm outperforms the state-of-the-art algorithms. Section V concludes the paper.

II. Problem Formulation

In this section, we describe our ABR streaming problem formulation. Let's assume a video divided into V chunks (segments), where every chunk is of length L seconds is encoded at one of $(N+1)$ quality levels. Let the n-th quality level of a chunk i be encoded at rate $r_{n,i}$. Let $X_{n,i}$ be the size of the i-th chunk when it is encoded at n-th quality level, $X_{n,i} = L * r_{n,i}$.

Let $Y_{n,i}$ be the size difference between the n and $(n-1)$-th quality levels, so $Y_{n,i} = X_{n,i} - X_{n-1,i}$, $n \geq 1$. However, $Y_{0,i} = X_{0,i}$. Moreover, let $I_{n,i}$ be an indicator variable that is equal to 1 if the i-th chunk can be fetched at least at the n-th quality level, and 0 otherwise, so $I_{n,i}$ is defined as follows:

$$\begin{cases} I_{n,i} = 1, & \text{if } i\text{-th chunk can be fetched at quality level} \geq n \\ I_{n,i} = 0, & \text{otherwise} \end{cases} \tag{1}$$

We will refer to $I_{n,i}$ by the decision variable of the n-th quality level. Let $Z_{n,i}$ be equal to $I_{n,i} \cdot Y_{n,i}$, *i.e.,* $Z_{n,i}$ is equal to how much is fetched out of $Y_{n,i}$. Since $I_{n,i} \in \{0,1\}$, $Z_{n,i}$ is $\in \{0, Y_{n,i}\}$. Since no chunk is totally skipped, we have $I_{0,i} = 1$, and $Z_{0,i} = Y_{0,i} \forall i$.

We assume an initial start-up delay of S time-units, and the video starts playing at this point. Since each chunk is of duration L, chunk i must be downloaded by time $S+(i-1)L$. If a chunk i cannot be downloaded by $S+(i-1)L$, a stall (*i.e.,* rebuffering) of $\alpha(i)$ seconds will occur, and the new deadline of every chunk $i' \geq i$ will increase by $\alpha(i)$ seconds, where $\alpha(i)$ is the more seconds required to fully download chunk i. As stalls continue occur during the video playback, the final deadline of every chunk i will be be $S+(i-1)L+\sum_{k=1}^{i} \alpha(k)$, *i.e.,* $deadline(i) = S + (i-1)L + \sum_{k=1}^{i} \alpha(k)$. During the stall, the video will pause until the chunk is fully downloaded since the buffer is running empty.

Let $X(i) = \sum_{n=0}^{N} Z_{n,i}$ be the decided size of the i-th chunk. Further, let $x(i,j)$ be how much out of $X(i)$ can be fetched at time j and $z_n(i,j)$ is what can be fetched out of $Z_{n,i}$ at time j. Moreover, let $B(j)$ be the predicted available bandwidth at time j. Also let B_m be the playback buffer size in time units, which indicates the maximum amount of video content that can be held in the playout buffer. We assume, without loss of generality, that the time unit is 1 second. When chunk i starts downloading, the buffer occupancy increases by L seconds (Recall that the chunk size is L seconds).

Now, we describe our problem formulation given available bandwidth prediction for W chunks ahead. Let's assume that the current W chunks are the chunks from i' to C where $C = i' + W - 1$, and the current time is j' (time from the start of the download). We refer to the total stall duration before this segment plus the start up delay by s, where $s = S + \sum_{k=1}^{i'-1} \alpha(k)$. Let $d(i)$ be the total stall duration encountered before the playback of the i-th chunk in the playback time of chunks in the segment $\{i', C\}$, *i.e.,* $d(i_1) > d(i_0)$ if $i_1 > i_0$. With these settings, we formulate an optimization problem that *(i)* minimizes the stall duration, *(ii)* maximizes the playback rate of the video accounting for diminishing returns with increase in chunk quality levels, and *(iii)* minimizes the quality changes between the neighboring chunks to ensure the perceived quality is smooth. We give a higher priority to *(i)* as compared to *(ii)* and *(iii)*, since stalls cause more quality-of-experience (QoE) degradation compared to playing back at a lower quality [7]. In particular, we consider an objective function that prefers pushing all the chunks to the n-th quality level over any other choice that might push the quality of some chunks to levels that are $> n$ with the cost of dropping the quality of some other chunks to levels $< n$.

To account for the above objectives, we choose the objective function to be: $\left(\sum_{n=1}^{N} \beta^n \sum_{i=i'}^{C} I_{n,i}\right) - \lambda d(C)$, where, $0 < \beta < 1$, and $\lambda >> 1$. The first part of the objective is a weighted sum of the level decision variables. More precisely it is a sum of the chunks obtained at least at the lowest quality plus β times the number of chunks obtained at least at the second quality level, and so on. More precisely, β should satisfy the following condition:

$$\sum_{i=i'}^{C} \sum_{k=n+1}^{N} \beta^k < \beta^n, \quad \text{for } n = 0, \cdots, N. \tag{2}$$

This choice of β implies diminishing returns with increasing quality levels. Fetching a chunk at quality n has more utility as compared to improving quality of the rest of the chunks beyond n. Thus, the use of β helps maximizing the minimum quality level among all chunks. This choice of β will not increase the quality of some chunks beyond the n-th quality level at the cost of dropping the quality of one or more chunks below the n-th quality level. The second term in the objective function is the the total stall duration. We assume $\lambda >> 1$. Therefore, avoidance of the stalls is given the highest priority. Due to these weights, the proposed algorithm will avoid stalls as the first priority and will not use the available bandwidth to increase the quality of some chunks at the expense of minimum chunk quality since lower levels are more preferable mirroring concave QoE function of the chunk rate. The optimization problem for the window $[i', C]$, where $C = i' + W - 1$ can be formulated as follows, where $I_{n,i}$ is the decision variable for the n-th quality level for chunk i.

$$\textbf{Maximize: } \left(\sum_{n=0}^{N} \left(\beta^n \sum_{i=i'}^{C} I_{n,i}\right) - \lambda \cdot d(C) \right) \tag{3}$$

subject to

$$I_{0,i} = 1, \forall i \tag{4}$$

$$\sum_{j=j'}^{deadline(i)} z_n(i,j) = Z_{n,i}, \forall i, n \tag{5}$$

$$I_{n,i} \leq I_{n-1,i}, \quad \forall i, n > 1 \tag{6}$$

$$Z_{n,i} = I_{n,i} \cdot Y_{n,i}, \quad \forall i, n \tag{7}$$

$$\sum_{n=0}^{N} \sum_{i=i'}^{C} z_n(i,j) \leq B(j), \forall j \tag{8}$$

$$\sum_{i, deadline(i) > t} \mathbf{I}\left(\sum_{j=1}^{t} \left(\sum_{n=0}^{N} z_n(i,j) \right) > 0 \right) L \leq B_m \ \forall t \tag{9}$$

$$z_n(i,j) \geq 0 \ , \forall i, j \tag{10}$$

$$z_n(i,j) = 0 \ \forall i, j > deadline(i) \tag{11}$$

$$I_{n,i} \in \{0,1\}, \forall i, n \tag{12}$$

$$d(i) \geq 0, deadline(i) = S + (i-1)L + \sum_{k=1}^{i'-1} \alpha(k) + d(i) \tag{13}$$

$$\text{Variables:} \quad z_n(i,j), I_{n,i}, Z_{n,i}, d(i) \quad \forall i = i', \cdots, C, \\ j = j', \cdots, deadline(C), \ n = 0, \cdots, N$$

Constraint (4) ensures that every chunk is fetched at least at lowest quality, *i.e.,* there are no skips. Constraint (5) defines the total amount fetched for chunk i. Constraint (6) ensures that if a chunk is not a candidate to $(n-1)$-th level, it won't be a candidate to the n-th level, so it should not be considered for the n-th quality level. Constraint (7) enforces $Z_{n,i}$ to be in $\{0, Y_{n,i}\}$. (8) imposes the available bandwidth constraint at each time slot j, and (9) imposes the playback buffer constraint so that the total playback in the buffer at any time does not exceed the buffer capacity B_m, where $\mathbf{I}(\cdot)$ is an indicator function. Therefore, as far as a chunk is partially downloaded, the buffer duration is increased by the duration of a chunk. Constraint (10) imposes the non-negativity of chunk download, and (11) imposes the deadline constraint, so no chunk is fetched after its deadline. (12) enforces the decision variables $I_{n,i}$ to belong to the discrete set $\{0,1\}$. Finally, Constraint (13) states that stall duration before any chunk is non-negative and defines the deadline of any chunk i.

The problem (3)-(13) is a non-convex optimization problem since the feasible set (the set of the decision variables) is discrete, and the problem contains non-convex constraint (9). However, we will show later that when the video is CBR encoded per quality level ($X_{n,i} = X_n, \forall i$, where X_n is a constant for the nth quality level), the proposed algorithm achieves the optimal solution to the problem (3)-(13).

Bandwidth Prediction: We note that the proposed formulation depends strongly on the available bandwidth prediction. In practice, perfect prediction cannot be non-causally available. There are multiple ways to obtain the prediction, including a crowd-sourced method to obtain historical data [18], [19]. Another approach may be to use a function of the past data rates obtained as a predictor of the future, as an example is to compute the exponential weighted moving average or the harmonic mean of the download time of the past η chunks to predict the future available bandwidth [20]. The weighted moving average smooths out the fluctuation of the available bandwidth measurements and the weight is given to the latest data. The harmonic mean method has been shown to mitigate the impact of large outliers due to network variations [16]. Therefore, in this paper, we consider using the harmonic mean of the download time of the last η chunks to predict the future available bandwidth. Given available bandwidth measurements of the last past η chunks, the predicted available bandwidth of the next segment is calculated as

$$\hat{B}(n+1) = \frac{\eta}{\frac{1}{\sum_{n-\eta}^{n} B(t)}}. \tag{14}$$

III. FastScan Algorithm

We describe the algorithm for adaptively streaming videos with qualities being decided based on the predicted available bandwidth, chunk deadlines, and buffer occupancy. The objective is to determine up to which quality level we need to fetch each chunk for the next window of chunks, such that the stall duration is minimized, and the proposed concave QoE metric in playback rate is maximized.

We will mainly describe the algorithm for a prediction window of W chunks. However, the available bandwidth prediction is updated after the download of each chunk using the harmonic mean based prediction technique. Thus, the quality decisions will be updated after each chunk's download based on the updated bandwidth prediction.

Algorithm 1 FastScan Algorithm

```
1:  **Input:** ; $X_n(i)\forall i, n$, $N$, $s$, $B_m$, $i'$, $j'$, $W$, $B(j)\forall j$,
2:  **Output:** $X(i)\forall i$: The maximum size in which chunk $i$ can be
    fetched, $I_n$: set contains the indices of the chunks that can be
    fetched up to $n$th quality level.
3:  **Initialization:**
4:  $C = i' + W - 1$
5:  $X(i) = 0$, $d(i) = 0$, $deadline(i) = s + (i-1)L$, $\forall i$
6:  $c(j) = \sum_{j'=1}^{j} B(j')$ available bandwidth up to time $j$, $\forall j$
7:  $bf(j) = 0$: buffer length at time $j$
8:  $t(i) = 0, \forall i$, first time slot in which chunk $i$ can be fetched
9:  $e(j) = B(j), \forall j$, remaining available bandwidth at time j after
    all non skipped chunk are fetched
10: **Lowest Quality level decision,** $n = 0$**:**
11: $d =$ **level-0-Forward**$(B, X, i', C, L, d, deadline,\ B_m, bf, j')$
12: $deadline(i) = (i-1)L + s + d(C)$
13: $d =$ **level-0-Backward**$(B,X_0,x,i',C,L,d,deadline,B_m,bf,j')$
14: $deadline(i) = (i-1)L + s + d(i)$
15: **Higher Quality Levels' decisions:**
16: **For every quality level** $n > 0$
17: $[t, a, e] =$ **level-n-forward**$(B,X,i',C,L,deadline,B_m,bf,j',I_0)$
18: $[X,I_n]$=**level-n-backward**$(B,X,X_n,i',C,L,deadline,$
    $B_m,bf,j',t,c,a,e)$
```

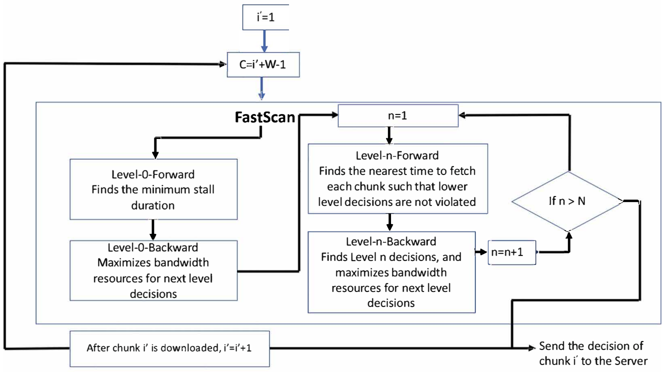
i′=1
C=i′+W-1
FastScan
Level-0-Forward
Finds the minimum stall duration
Level-0-Backward
Maximizes bandwidth resources for next level decisions
n=1
Level-n-Forward
Finds the nearest time to fetch each chunk such that lower level decisions are not violated
Level-n-Backward
Finds Level n decisions, and maximizes bandwidth resources for next level decisions
n=n+1
If n > N
After chunk i′ is downloaded, i′=i′+1
Send the decision of chunk i′ to the Server

Fig. 1. Flowchart Diagram of FastScan Algorithm

As shown in Fig. 1, FastScan algorithm is called after the download of every chunk to make the quality decisions of the next W chunks starting from chunk i'. Therefore, the algorithm is called in a sliding window manner in which the previous decisions of some chunks may change on the fly and an additional chunk is considered in every call. Before we describe the algorithm in detail, we introduce the high level idea of the algorithm as explained in Fig. 1. At a high level, FastScan performs forward followed by backward scans at each quality level. The scans simulate fetching chunks in forward and reverse order, respectively. The Level-0 forward and backward scan's objective is to find the minimum stall duration and maximize the resources of the next level decision as it will be explained in more detail later. However, starting from level 1, the job of every Level-n forward and backward scans is to maximize the number of candidate chunks to the n-th quality level without violating the lower level decisions.

The algorithm is summarized in Algorithm 1, and it works as follows. Given the size of the lowest quality level for every chunk, chunks' deadlines, predicted available bandwidth, and buffer size, FastScan performs a forward scan followed by a backward scan by calling Level-0 Forward and Level-0 Backward algorithms (Algorithms 2 and 3) assuming that all chunks need to be fetched at the lowest quality level (0th quality level). Based on the first pass of the forward and backward algorithms, decisions of the stall duration, stall pattern and the final deadline of every chunk such that the stall duration is minimized and the bandwidth resources to every chunk are maximized are found. Consequently, the Level-n Forward and Backward ($n = 1, \cdots, N$) algorithms are performed in order to find the candidate chunks to the nth quality level. The final quality decision for every chunk in the current window of chunks will be the return of the backward algorithm's call of the highest quality level. The algorithm is re-run after the download of every chunk, so it keeps adjusting for the prediction error and changing decisions if necessary on the fly. In order to reduce the complexity, the algorithm uses the cumulative predicted available bandwidth of the jth second ($c(j)$, line 5), so the algorithm avoids having double loops summing the available bandwidth over multiple time slots when making the decision per chunk.

Level-0-forward algorithm finds the minimum stall time such that all chunks are fetched at the lowest quality level, then the algorithm assumes that all stalls can be brought to the beginning of the current segment by setting the deadline of every chunk in this segment to the value defined by line 11. Level-0-Backward algorithm (line 12) runs after that, and its job is to check if all stalls can indeed be brought to the beginning of this window of chunks, if not, the algorithm will find the minimum number of later stalls. The goal of the backward algorithm is to maximize the number of candidate chunks to the next level since if all stalls can be shifted to their earliest, since in this case all chunks have higher chance to increase their qualities. Therefore, the deadline that is found after running Level-0-backward algorithm (line 13) is final. Finally, the Level-n Forward and Backward algorithms are run per level to find the chunks that can still be fetched at higher quality levels. We will now describe the forward and backward algorithms.

Algorithm 2 Level-0 Forward Algorithm

```
 1: **Output:** $d(i)$: stall time of chunk $i$.
 2: $j = j'$, $i = i'$
 3: **while** chunk $C$ is not fetched yet **do**
 4:     **if** $(i > 1$ and $d(i) < d(i-1))$ **then**
 5:         $d(i) = d(i-1)$, $deadline(i) = (i-1)L + s + d(i)$
 6:     **end if**
 7:     **if** $(bf(j) = Bm)$, **then** $j = j + 1$
 8:     $fetched = min(B(j), X(i))$
 9:     $B(j) = B(j) - fetched$, $X(i) = X(i) - fetched$
10:     **if** $X(i) > 0$, **then** $bf(j) = bf(j) + L$
11:     **if** $X(i) = 0$ and $j <= deadline(i)$ **then**
12:         i=i+1
13:     **else if** $X(i) = 0$ and $j > deadline(i)$ **then**
14:         $d(i) = d(i) + j - deadline(i)$
15:         $i = i + 1$
16:     **end if**
17:     **if** $B(j) = 0$, **then** $j = j + 1$
18: **end while**
```

Algorithm 3 Level-0 Backward Algorithm

```
 1: **Output:** $d_f(i)$: final stall duration of chunk $i$.
 2: **Initilization:**
 3: $i = C$, $j = deadline(C)$
 4: $d_f(i) = d(i)$, and $deadline(i) = (i-1)L + s + d_f(i)$ $\forall i$
 5: **while** $(j > (j'-1)$ and $i > (i'-1))$ **do**
 6:     **if** $(i < C)$ **then**
 7:         $d_f(i) = d(i) - (d(i+1) - d_f(i+1))$
 8:         $deadline(i) = (i-1)L + s + d_f(i)$
 9:     **end if**
10:     **if** $j <= deadline(i)$ **then**
11:         **if** $(bf(deadline(i)) = B_m)$ **then**
12:             $d_f(i) = d(i) - 1$
13:             $j = j - 1$
14:         **else**
15:             $fetched = min(B(j), X_0(i))$
16:             $X_0(i) = X_0(i) + fetched$, $B(j) = B(j) - fetched$
17:             **if**$(X(i) > 0)$, **then** $bf(j) = bf(j) + L$
18:             **if**$(X_0(i) = 0)$, **then** $i = i - 1$
19:             **if**$(B(j) = 0)$, **then** $j = j - 1$
20:         **end if**
21:     **else**
22:         $j = j - 1$
23:     **end if**
24: **end while**
```

Level-0 Forward algorithm: To explain Level-0 Forward algorithm, let's assume that the current W chunks that need to be fetched are the chunks from i' to C where $C = i' + W - 1$. The algorithm simulates fetching these chunks in order at the lowest quality level with an assumption that the stall duration of each chunk in this segment is zero, *i.e.,* $d(i') = \cdots = d(C) = 0$, $\forall i \in \{i', C\}$. Therefore, the initial deadline of every chunk i such that $i' \leq i \leq C$ is $(i-1)L + s$, where $s = S + \sum_{k=1}^{i'-1} \alpha(k)$ is the total stall duration before the current segment. Chunks are fetched in order and if a chunk i can't be completely downloaded before its initial deadline, the additional time spent in fetching this chunk is added to $d(i'')$ for every $i \leq i'' \leq C$ (lines 13-16) since there has to be an additional stall in order to fetch these chunks. Therefore, level-0 Forward algorithm finds the total stall duration of this segment of chunks and the deadline of the last chunk ($d(C)$, and $deadline(C)$) such that all chunks are guaranteed to be fetched at least at the lowest quality level. Two things to note, first, Level-0-forward algorithm gives a guarantee that all chunks from i' to C can be fetched before the time slot $deadline(C)$ at least at the lowest quality level with the minimum stall duration according to the predicted available bandwidth. Second, forward algorithm may lead to stalls in the middle of the segment which can hurt the performance by having less opportunity to fetch later chunks at higher quality levels. To avoid this situation, we run the Level-0 Backward algorithm right after the Level-0 forward algorithm to move stalls earlier helping more chunks to possibly be fetched at higher quality.

Level-0 Backward algorithm: Given the predicted available bandwidth, the forward algorithm decisions, and the buffer size, Level-0 Backward algorithm (Algorithm 3) simulates fetching chunks at **level-0** quality in reverse order starting at the deadline of the last chunk ($j = deadline(C) = (C-1)L + s + d(C)$). Therefore, the Level-0 Backward algorithm simulates fetching chunks close to their deadlines. The backward algorithm's objective is to move stalls to their earliest time, *i.e.,* as early as possible. In fact, if the buffer is infinite, all stalls can be brought to the beginning of the current segment, but if the buffer is finite, that may not always be possible. There will be one scenario that leads to changing the initial deadline and having a stall that is not brought to the very beginning of the current segment which is the buffer constraint violation (lines 11-13). If fetching a chunk i results in a buffer constraint violation, the algorithm decrements the deadline of chunk i by 1 and checks if the violation can be removed. This decrement will continue until the buffer constraint is no longer violated. The aim of the Level-0 Backward algorithm is to provide the deadlines of the different video chunks such that the fetching of the video minimizes the stall duration. Further, Level-0 Backward algorithm aims to bring all the stalls to their earliest possible time, since this provides all chunks more time to increase their qualities without violating $deadline(C)$. Therefore, later chunks have a higher chance of increasing their quality.

Once **Level-0** decisions are found, the **Level-n** forward, and the **Level-n** backward algorithms (Algorithm 4, and 5) are run per level. The key difference of **Level-n** forward and **Level-n** backward algorithms as compared to **Level-0** forward and backward algorithms is that the deadlines are fixed and can't be changed. Therefore, if a chunk can't be fetched at nth quality level with its current deadline, it is not considered as a candidate to that quality level.

Level-n forward algorithm simulates fetching chunks in order, where chunks are fetched according to the $(n-1)$th level decision. The main objective of **Level-n** forward algorithm is to determine the lower deadline of every chunk i in which chunk i can't be fetched earlier ($t(i)$, lines (10-11)); otherwise, the lower level decisions are violated. As we will see next, the lower and upper deadlines of every chunk are the key components that **Level-n** backward algorithm use to make a decision of which chunk can be fetched at the nth quality level.

Level-n backward algorithm takes as input the lower and the upper deadlines of every chunk and finds the set of chunks that can be pushed to the nth quality level. If pushing a chunk to nth quality level is not possible without violating its lower and upper deadlines, the chunk will not be a candidate to the nth quality level. Therefore, we clearly see that the algorithm prioritizes pushing the maximum number of chunks to the $(n-1)$-th quality level over fetching some at the n-th quality level at the cost of reducing the quality of some others below the $(n-1)$th quality level. This will prioritize horizontal over vertical scanning which reduces the quality switching that is spanning more than one quality-level.

The backward algorithm considers if the chunk can be a candidate for the nth quality level. In order to decide this, the algorithm determines if there is enough available bandwidth and the buffer is not full (line 17-21). If chunk i was a candidate to the $(n-1)$-th quality level and is not selected to be fetched at the n-th quality level. If the chunk was a candidate to the $(n-1)$-th quality level and is not selected to be fetched at the nth quality level, this could be because of one of the two scenarios mentioned next. The first is the violation of the **buffer** capacity, where the chunk could not be placed in the buffer before its deadline (line 7). The second is the **available bandwidth** constraint violation where the remaining available bandwidth is not enough for downloading the chunk at the n-th quality level (lines 14-16).

For buffer capacity violation, we first note that there could be a chunk $i'' > i$, which if it is not fetched at the n-th quality level, chunk i could have been an n-th level candidate. However, **Level-n** backward algorithm decides to not consider i (line 7). We note that since there is a buffer capacity violation, one of the chunks $\geq i$ must be skipped (not considered for the n-th level). The reason for choosing to not consider (skip) chunk i rather than a later one is that i is the closest one to its deadline. Therefore, i is not a better candidate to the next quality level than any of the later chunks.

In the second case of deadline/available bandwidth violation, **Level-n** backward algorithm decides to skip chunks up to i since there is not enough available bandwidth. As before, an equal number of chunks need to be skipped (for CBR encoding), skipping earlier ones is better because it helps increasing the potential of getting later chunks at higher quality levels.

Finally, giving higher priority to later chunks reduces the effect of the inaccurate available bandwidth estimation. In other words, by giving priority to later chunks, if the real available bandwidth turns to be lower than the predicted one, the prediction error effect is minimized. Moreover, to handle prediction error, a lower buffer threshold can be set, so if the buffer is running lower than this threshold, the level decision that was made by our streaming algorithm for the next chunk is reduced by 1 quality level (except if the chunk is already at lowest quality).

Algorithm 4 **Level-n** Forward Algorithm

```
 1: Output: $t(i)$, $a(i)$, $e(j)$
 2: $j = 1$, $k = 1$
 3: while $j \leq deadline(C)$ and $k \leq max(I_0)$ (last chunk to fetch) do
 4:     $i = I(k)$
 5:     if $i = 0$ then  $k = k + 1$
 6:     if $j \leq deadline(i)$ then
 7:         if  $(bf(j) = B_m)$ then  $j = j + 1$
 8:         $fetched = min(B(j), X(i))$
 9:         if $j$ is the first time chunk $i$ is fetched then
10:             $t(i) = j$, $a(i) = fetched$
11:         end if
12:         $B(j) = B(j) - fetched$, $e(j) = B(j)$, $X(i) = X(i) - fetched$
13:         if  $X(i) > 0$ then  $bf(j) = bf(j) + L$
14:         if  $X(i) = 0$ then  $k = k + 1$
15:         if  $B(j) = 0$ then  $j = j + 1$
16:     else
17:         $k = k + 1$
18:     end if
19: end while
```

Complexity Analysis: We note that the initialization step sums the variables over time, and is thus O(W) complexity. The backward and forward algorithms are run once per quality level. For each run of backward/forward algorithm, there is a while loop, and within the loop, the complexity is O(1). Since the while loop runs at most $W + deadline(i' + W - 1) + 1$ times, the overall complexity of the proposed algorithm is O(NW). In order to decrease the complexity, cumulative available bandwidth for every time slot j, $c(j)$ is used to avoid summing over the available bandwidth in the backward and the forward loops.

Optimality of FastScan Algorithm for CBR encoded videos: We note that the proposed algorithm is a variant of the algorithm proposed for SVC encoded videos in [15]. Adapting the results in [15], [21], the optimality of the proposed algorithm in solving (3)-(13) follows. Therefore, in the offline case in which the bandwidth is perfectly predicted for the whole period of the video, and $W = V$, the obtained quality is the offline optimal. The result is summarized in the following theorem, and the proof is omitted since it is an adaptation of the result in [15].

Theorem 1. *Up to a given quality level $M, M \geq 0$, if $(I^*_{m,i}, d^*(i))$ are the m-th quality level decision variable $m \leq M$ and stall duration of chunk i that are found by using FastScan algorithm, and $I'_{m,i}, d'(i)$ are the decision variable and the stall duration that are found using any other feasible algorithm, then the following holds for any $0 < \beta < 1$, satisfies (2).*

$$\sum_{m=0}^{M} \beta^m \sum_{i=i'}^{C} I'_{m,i} - \lambda d'(C) \leq \sum_{m=0}^{M} \beta^m \sum_{i=i'}^{C} I^*_{m,i} - \lambda d^*(C). \tag{15}$$

Algorithm 5 Level-n Backward Algorithm

```
Initialization:
$i = C$, $j = deadline(C)$
while ($j > 0$ and $i > 0$) do
    if $j <= deadline(i)$ then
        if $(bf(deadline(i)) = B_m)$ then $i = i - 1$
        if $j$ is the first time to fetch chunk $i$ from back then
            if $(t(i) = 0)$ then
                $rem1 = c(j) - c(1) + e(1)$, $rem2 = rem1$
            else
                $rem2 = c(j) - c(t(i))$, $rem1 = rem2 + e(t(i)) + a(i)$
            end if
            if $(rem1 < X_n(i))$ then
                if $(X(i) > 0)$ then $X_n(i) = X(i)$ else $i = i - 1$
            else
                if $(rem2 < X_n(i))$ and $rem1 \geq X_n(i))$ then
                    $e(t(i)) = e(t(i)) + rem1 - X_n$
                end if
                $X(i) = X_n(i)$, $I_n \leftarrow I_n \cup i$
            end if
        end if
        $fetched = min(B(j), X_n(i))$
        $B(j) = B(j) - fetched$, $X_n(i) = X_n(i) - fetched$
        if $(X_n(i) > 0)$ then $bf(j) = bf(j) + L$
        if $(X_n(i) = 0)$ then $i = i - 1$
        if $(B(j) = 0)$ then $j = j - 1$
    else
        $j = j - 1$
    end if
end while
```

In other words, FastScan Algorithm finds the optimal solution to the optimization problem (3-13) when $0 < \beta < 1$*, (2) holds, and* $\lambda \gg 1$.

Discussions: We note that the optimization problem in this paper is a combinatorial optimization problem, with discrete constraints. Many known combinatorial optimization problems are NP hard (*e.g.,* Knapsack problem). We also note that the combinatorial optimization problem is not necessarily NP hard. For example, matching problem is one of the combinatorial problem that is not NP-hard. The NP-hard Knapsack problem optimizes a linear function of integer variables with a single linear constraint. The proposed problem has multiple constraints, which intuitively makes it harder. However, the structure in the problem considered in this paper allows us to find optimal algorithm which has linear time complexity. We note that a straightforward greedy level-by-level optimization will not be optimal. This is because the decisions at the higher levels depend on the decisions at the lower levels. Thus, the decisions at the lower levels must leave large available bandwidth for fetching the incremental content of higher quality. The algorithm in this paper helps account for the connections for different layers and is optimal for the proposed problem. Thus, the diminishing returns in increasing quality levels make both a practical motivation and leads to a computationally simple optimal algorithm for CBR encoded videos.

IV. Implementation

In this section, we describe our implementation of the proposed approach in the dash.js framework. Our implementation is based on the dash.js master branch (v1.2.0 release) with the modifications proposed by [7].

A. Setup

Our choice of dash.js framework is motivated by the fact that dash.js is a reference open-source implementation for the MPEG-DASH standard and is actively supported by leading industry participants [7]. The system architecture of dash.js is shown in Fig. 2. Briefly, dash.js abstracts the high-level video streaming functionalities such as the rate adaptation logic on top of the low-level DASH standard related components. The main class that is responsible for the rate adaptation techniques is AbrController. This class contains the core bitrate adaptation logic. In the original dash.js implementation, a rule-based decision logic is implemented to find the bitrate. Specifically, DownloadRatioRule selects bitrate based on the download ratio (play time of last chunk divided by its download time). On the other hand, InsufficientBufferRule chooses bitrate depending on whether the buffer level has reached a lower limit recently to avoid re-buffers. Priorities are assigned to each rule to resolve conflicts and make final bitrate decisions [7]. Further, we use the modifications to dash.js that was proposed by [7] which allow it to work with prediction based adaptation algorithms. For details on such modifications, the reader is referred to [7].

We compare our algorithm (FastScan) with the FastMPC, Pensieve, Bola, and the other existing buffer and prediction based algorithms which were described by [7]. We set the maximum buffer size to 1 minute for all considered algorithms $B_m = 1$ minute. We briefly describe the comparable approaches here, and for more information about these algorithms, see [7].

• FastMPC: The FastMPC implementation has a static table that is used to index control decisions. It uses the harmonic mean to predict the available bandwidth. We use default setting proposed by [7]. The prediction window is of size 5 chunks (i.e. look-ahead horizon $h = 5$) with throughput predictions using harmonic mean of past 5 chunks. We use 100 bins for throughput prediction and 100 bins for buffer level.

• Pensieve: Pensieve is a Reinforcement Learning (RL) based rate adaptation scheme in which an RL agent is trained offline using real bandwidth traces to make quality decision per chunk. Once the RL is trained, it is used to make quality decisions for streaming video chunks. Pensieve uses the QoE metric that is proposed in [7] as its reward function. we run Pensieve using the trained model provided by [14].

• FESTIVE: [16]: It calculates efficiency score depending on throughput predictions using harmonic mean of the past 5 chunks, as well as a stability score as a function of the bitrate switches in the past 5 chunks. The bitrate is chosen to minimize stability score+α × efficiency score, where $\alpha = 12$.

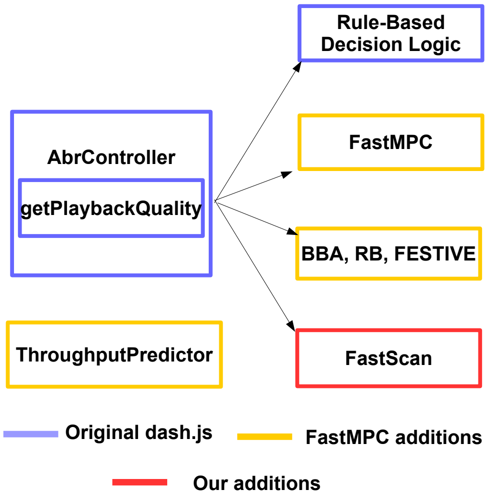

Fig. 2. dash.js code structure, FastMPC modifications [7], and our modifications

- BBA: Proposed by [11], it adjusts the streaming quality based on the playback buffer occupancy. Specifically, it is configured with lower and upper buffer thresholds (reservoir and cushion). If the buffer occupancy is lower (higher) than the lower (higher) threshold, chunks are fetched at the lowest (highest) quality; if the buffer occupancy lies in between, the buffer-rate relationship is linear. we set reservoir $r = 10s$ and cushion $c = 30s$.

- RB: The bitrate of the next chunk is picked as the maximum available bitrate which is less than throughput prediction using harmonic mean of past 5 chunks.

- Bola: The original dash.js implementation that adopts a rule-based bitrate decision logic as shown in Fig. 2.

To make the comparison fair, we have the following settings for our algorithm: the prediction window is 5 chunks ($W = 5$), we use the time corresponding to the download of the last 5 chunks to predict the future available bandwidth ($\eta = 5$). Moreover, we use the nominal chunk size of all levels (not the exact chunk size) when we run the forward-backward algorithms. Therefore, we consider the performance when we have knowledge about the mean chunk sizes. We believe knowing the exact size will further improve the performance of our algorithm. Finally, we set the lower buffer threshold to $5s$ which is a little higher than playback time of 1 chunk. In other words, if the buffer does not contain a chunk, and the FastScan decision about the next chunk size is higher than the 1-st quality level, we reduce the quality decision that is made by FastScan algorithm by 1 quality level.

Video parameters: We use the Envivio video from DASH-264 JavaScript reference client test page [22], which is 260s long, consisting of 65 4s chunks ($L = 4s$). The video is encoded by H.264/MPEG-4 AVC codec into 5 different quality levels, and the nominal bitrate of the levels are as follows: $R = \{0.338\text{Mbps}, 0.583\text{Mbps}, 0.959\text{Mbps}, 1.898\text{Mbps}, 2.806\text{Mbps}\}$. The actual encoding rates of the chunks at any of the 5 different levels are different from the nominal rates since the video is encoded in variable bit rate (VBR). The min, mean, and max chunk size of every quality level in Mega Bytes (MB) are shown in Table I (chunk size of level i is equal to $L * R(i)$). The table clearly shows the variability of the chunk sizes of every level.

TABLE I
AVC ENCODING CHUNK SIZES IN MB OF THE ENVIVIO VIDEO USED IN OUR EVALUATION

Quality level	level-0	level-1	level-2	level-3	level-4
Min	0.0433	0.0786	0.1265	0.2213	0.3205
Mean	0.1693	0.2916	0.4795	0.949	1.403
Max	0.2342	0.3855	0.6217	1.286	1.918

Bandwidth Traces: For available bandwidth traces, we used traces from three sets of public dataset. The first dataset consists of continuous 1-second measurement of throughput of a moving device in Telenor's mobile network in Norway [18]. The dataset is post-processed by [7] to make short traces that matches the video length. We avoid the traces in which the quality decision is trivial (*i.e.,* the very low traces in which the decision is to fetch all chunks at the lowest quality level, and the very high average traces, in which the decision is to fetch chunks at highest quality level). The second dataset is the FCC dataset [23], which consists of more than 1 million sets of throughput measurements. The last dataset consists of 40 LTE traces collected in Belguim [24]. Since LTE traces have high bandwidth values, we scale every trace by a factor of $1/5$ to create more challenging scenarios.

B. Evaluation

In this section, we compare our approach against several prior approaches. Our basic experiment setup consists of two computers (Ubuntu 12.04 LTS) emulating a video server and a client. The video client is a GoogleChrome web browser with V8 JavaScript engine while the video server is a simple HTTP server based on node.js (version 0.10.32). We use the dummynet [25] tool to throttle the throughput of the link between two computers according to the throughput traces employed.

The results are shown in Figures 3 – 5. Fig. 3-a shows the CDF of the normalized QoE. The QoE metric is our objective function when $\beta = 0.1$ and $\lambda = 10$. The normalization is with respect to FastScan's QoE. We clearly see that in the HSDPA dataset (Norway dataset), our algorithm acheives the highest QoE for about 99% of the bandwidth traces, and the gain is significant in some of the traces. To get more insights we plot in Fig. 3-b the probability mass function of the number of chunks fetched at the different quality levels (e.g "0"= the lowest quality level, and "4"= the highest quality level).

The percentage of the chunks that are downloaded at the poorest quality (0-th quality level) and the total re-buffering (stall) duration for each algorithm among all traces is displayed on the top of Fig. 3-b. We first see that FastScan outperforms most of the baseline algorithms in terms of avoiding

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2019.2914388, IEEE Transactions on Circuits and Systems for Video Technology

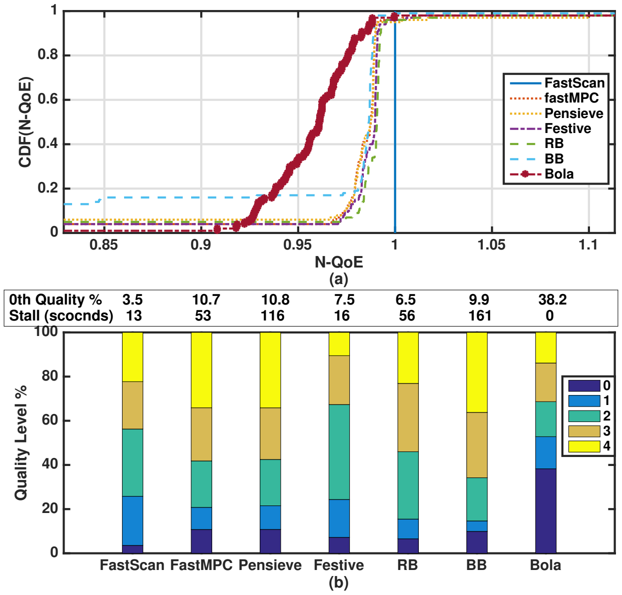

Fig. 3. Comparison with other approaches: Norway Dataset

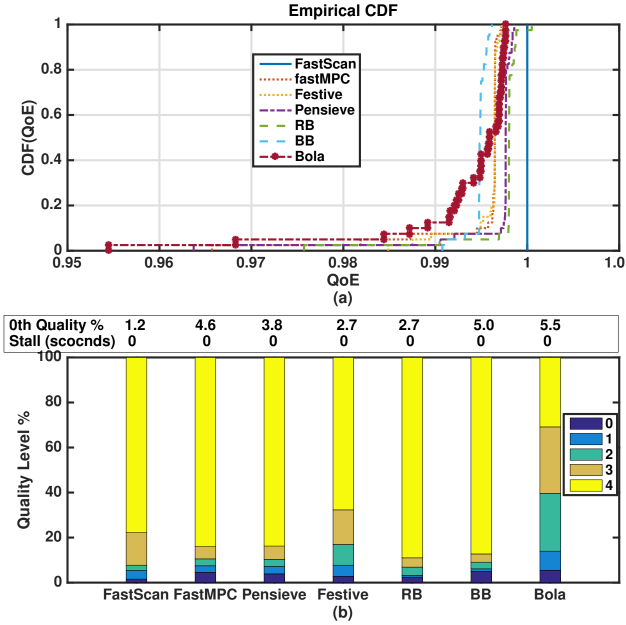

Fig. 5. Comparison with other approaches: LTE dataset

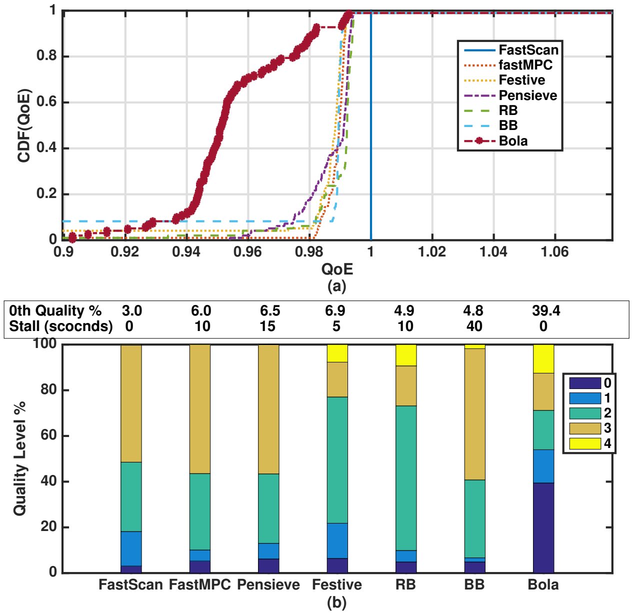

Fig. 4. Comparison with other approaches: FCC dataset

stalls (re-buffering durations). For instance, over 100 available bandwidth traces, FastScan runs into 13 seconds of buffering time (stalls). On the other hand, FastMPC, the algorithm that achieves the closest average playback to FastScan runs into 53 seconds of the stall time which is about 4 times higher than the FastScan's stall duration. We clearly see that FastScan maintains a very low stall duration and pushes more chunks to higher quality levels. FastScan fetches only 3.4% of the chunks at the poorest (0-th) quality level which is about half of the Festive's percentage and one third of FastMPC and Pensieve. FastMPC and Pensieve fetch more chunks at the highest quality level than FastScan, but that comes at the cost of fetching more chunks at the poorest quality level and running into more stalls. However, since the considered QoE metric is a concave function with respect to the playback rate, pushing more chunks from 0-th to the 1st quality level achieves higher QoE than fetching more chunks at the highest quality level at the cost of dropping the quality of some chunks to the 0-th quality level. Therefore, FastScan achieves higher QoE since it processes quality levels in order and maximizes the number of candidate chunks to the n-th quality level before considering the $(n+1)$ quality level. It worth mentioning that Bola achieves the minimum stall duration (0 seconds), but that comes at the cost of achieving significantly lower QoE in most of the bandwidth traces and the highest percentage of chunks that are fetched at $0-th$ quality level (38.2%). Finally, FastScan incorporates available bandwidth prediction and the deadline of the chunks into its decisions, prioritizes the later chunks, and re-considers the decisions periodically (after the download of every chunk). These properties help FastScan be adaptive to different available bandwidth regimes and variations in the available bandwidth profiles.

Fig. 4 compares the results of FastScan with the other algorithms in FCC dataset, and Fig. 5 compares the results of FastScan with the other algorithms in LTE datset. In both datasets, we see qualitatively similar results to what we have described in HSDPA dataset. In the two set of traces, our algorithm achieves highest QoE in more than 99% of the traces. Moreover, our algorithm is able to manage the trade-off between pushing more chunks to higher quality levels, and avoid running into stalls. It maintains a stall duration that is as low as the one achieved by the most conservative algorithm and high average playback as high as the one that is achieved by the most optimistic algorithm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2019.2914388, IEEE Transactions on Circuits and Systems for Video Technology

In conclusion, the proposed algorithm (FastScan) significantly outperforms the considered baselines in terms of avoiding stalls and pushing more chunks to their highest quality levels. Incorporating the predicted bandwidth and the deadline of the chunks into its decisions, prioritizing the later chunks, and re-considering the decisions after the download of every chunk make FastScan adaptive to different available bandwidth regimes and variations in the available bandwidth profiles.

V. Conclusion

This work considers the problem of adaptive bit-rate video streaming, which optimizes a novel QoE metric that models a combination of the three objectives of minimizing the stall/skip duration of the video, maximizing a concave function of the playback quality averaged over the chunks, and minimizing the number of quality switches. A low-complexity algorithm is proposed to solve the optimization problem. Due to the structure of the proposed QoE, the proposed algorithm can be shown to be optimal under some assumptions. Extensive evaluations with real videos and available bandwidth traces of a public dataset reveal the practicality and the robustness of the proposed scheme and demonstrate its significant performance improvement as compared to the state-of-the-art ABR streaming algorithms. This work can be extended for the case when client hosts are equipped with multiple network interfaces as in [26], [27], [28].

References

[1] M. Harel, "What is Driving the Growth of Mobile Video?" Jun. 2017.

[2] "Video as a Basic Service of LTE Networks: Mobile vMOS Defining Network Requirements," *http://www.huawei.com/minisite/4-5g/en/industryjsdc-j.html*, July 2015.

[3] "MPEG-DASH," https://www.iso.org/standard/65274.html, 2014.

[4] "Apple HTTP Live Streaming," https://goo.gl/6yYWg.

[5] "Microsoft Smooth Streaming," http://goo.gl/TQHWL.

[6] "Adobe HTTP Dynamic Streaming," http://goo.gl/IZWE8d.

[7] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015.

[8] K. Miller, D. Bethanabhotla, G. Caire, and A. Wolisz, "A Control-Theoretic Approach to Adaptive Video Streaming in Dense Wireless Networks," *IEEE Trans. Multimedia*, vol. 17, no. 8, 2015.

[9] D. Jarnikov and T. Özçelebi, "Client intelligence for adaptive streaming solutions," *Sig. Proc.: Image Comm.*, vol. 26, no. 7, 2011.

[10] M. Claeys, S. Latré, J. Famaey, and F. D. Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client," *IEEE Communications Letters*, vol. 18, no. 4, 2014.

[11] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM SIGCOMM*, 2014.

[12] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang, "C3: Internet-Scale Control Plane for Video Quality Optimization," in *Proc. NSDI*, 2015.

[13] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction," in *Proc. ACM SIGCOMM*, 2016.

[14] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.

[15] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen, "Lbp: Robust rate adaptation algorithm for svc video streaming," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1633–1645, 2018.

[16] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 97–108.

[17] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE, 2016, pp. 1–9.

[18] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.

[19] J. Hao, R. Zimmermann, and H. Ma, "GTube: geo-predictive video streaming over HTTP in mobile environments," in *Proc. ACM MMSys*, 2014.

[20] Y.-C. Chen, D. Towsley, and R. Khalili, "Msplayer: Multi-source and multi-path video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2198–2206, 2016.

[21] A. Elgabli and V. Aggarwal, "Deadline and buffer constrained knapsack problem," *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[22] "dash.js JavaScript Reference Client," https://reference.dashif.org/dash.js/.

[23] "Federal Communications Commission. 2016. Raw Data - Measuring Broadband America. (2016)." https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.

[24] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[25] "The dummynet project," http://info.iet.unipi.it/~luigi/dummynet/, 2017, online; accessed 14 July 2017.

[26] A. Elgabli, K. Liu, and V. Aggarwal, "Optimized preference-aware multi-path video streaming with scalable video coding," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2019.

[27] A. Elgabli, M. Felemban, and V. Aggarwal, "Groupcast: Preference-aware cooperative video streaming with scalable video coding," *IEEE/ACM Transactions on Networking*, 2019.

[28] ——, "Giantclient: Video hotspot for multi-user streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2018.

Anis Elgabli received M.S. and Ph.D. degrees in 2015 and 2018, respectively from Purdue University, IN, USA, all in Electrical and Computer Engineering. He is currently a Postdoctoral Researcher at the University of Oulu, Oulu Finland. His research interest is in applying optimization techniques in networking and communication systems. He was the recipient of the 2018 Infocom Workshop HotPOST Best Paper Award.

Vaneet Aggarwal (S'08 - M'11 - SM'15) received the B.Tech. degree in 2005 from the Indian Institute of Technology, Kanpur, India, and the M.A. and Ph.D. degrees in 2007 and 2010, respectively from Princeton University, Princeton, NJ, USA, all in Electrical Engineering. He was a Senior Member of the Technical Staff Research with AT&T Labs Research, Bedminster, NJ, USA, from 2010 to 2014. He is currently an Associate Professor with Purdue University, West Lafayette, IN, USA. His current research interests are in communications and networking, video streaming, cloud computing, and machine learning. He received Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the 2017 Jack Neubauer Memorial Award recognizing the Best Systems Paper published in the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the 2018 Infocom Workshop HotPOST Best Paper Award.