BOLA360: Near-optimal View and Bitrate Adaptation for 360-degree Video Streaming

Ali Zeynali[*] Mahsa Sahebdel[†] Mohammad Hajiesmaili[‡] Ramesh K. Sitaraman[§]

October 1, 2024

Abstract

Recent advances in omnidirectional cameras and AR/VR headsets have spurred the adoption of 360° videos, which are widely believed to be the future of online video streaming. 360° videos allow users to wear a head-mounted display (HMD) and experience the video as if they are physically present in the scene. Streaming high-quality 360° videos at scale is an unsolved problem that is more challenging than traditional (2D) video delivery. The data rate required to stream 360° videos is an order of magnitude more than traditional videos. Further, the penalty for rebuffering events where the video freezes or displays a blank screen is more severe as it may cause cybersickness. We propose an online adaptive bitrate (ABR) algorithm for 360° videos called `BOLA360` that runs inside the client's video player and orchestrates the download of video tiles from the server to maximize the quality-of-experience (QoE) of the user. `BOLA360` conserves bandwidth by downloading only those video tiles that are likely to fall within the field-of-view (FOV) of the user. In addition, `BOLA360` continually adapts the bitrate of the downloaded video tiles so as to enable a smooth playback without rebuffering. We prove that `BOLA360` is near-optimal with respect to an optimal offline algorithm that maximizes QoE. Further, we evaluate `BOLA360` on a wide range of network and user head movement profiles and show that it provides 6% to 110% improvements to the QoE of state-of-the-art algorithms. While ABR algorithms for traditional (2D) videos have been well-studied over the last decade, our work is the first ABR algorithm for 360° videos with *both* theoretical and empirical guarantees on its performance.

[*]University of Massachusetts Amherst. Email: `azeynali@cs.umass.edu`.
[†]University of Massachusetts Amherst. Email: `msahebdelala@umass.edu`.
[‡]University of Massachusetts Amherst. Email: `hajiesmaili@cs.umass.edu`.
[§]University of Massachusetts Amherst & Akamai Technologies. Email: `ramesh@cs.umass.edu`.

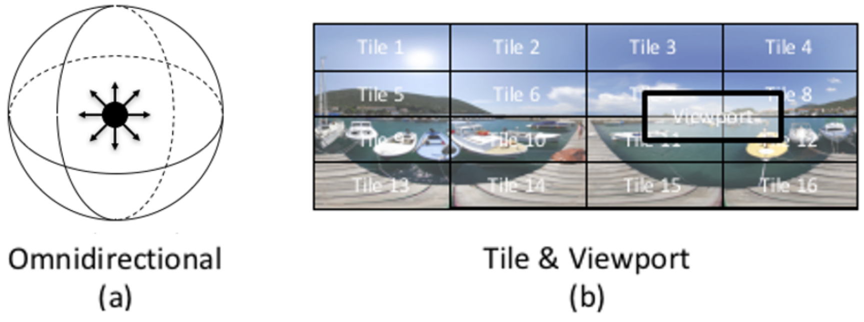


Figure 1: (a) Users watch 360° videos by moving their viewport to point to any direction in the enclosing sphere (b) each frame of the 360° video is broken up into tile frames [5].

1 Introduction

With recent advancements in omnidirectional cameras and AR/VR headsets, users can enjoy 360° media like YouTube 360 [1], virtual and augmented reality applications [2, 3].Users either wear a head-mounted display (HMD) or use a device that allows them to change their viewport and field-of-view (`FOV`)[1] when watching a 360° video (see Figure 1). For instance, a user watching World Cup soccer as a 360° video can wear an HMD and watch the game by changing their head position as if they were actually in the stadium.

The rapid increase in the popularity of 360° videos is partly driven by the wide availability of VR headsets that has grown more than five-fold in the past five years to reach nearly 100 million units in use [4]. A second trend driving the popularity of 360° videos is the wide availability of omnidirectional cameras that make it easy to create 360° video content. While the promise of providing an immersive experience has made 360° videos the holy grail of internet video streaming [5], providing a high quality-of-experience to users while *delivering those videos at scale over the internet* is a major unsolved problem and is the main motivation of our work.

Tiled video delivery. A common approach to deliver 360° video from server to users (i.e., client) is to divide the entire 360° video into same duration chunks of length δ. Then, each chunk is spatially split into a set of tiles to fully cover the viewing sphere of the user (see Figure 1). Each tile is encoded in multiple bitrates (i.e., resolutions) so that the quality of the tiles sent to the user can be adapted to the available bandwidth between the server and the client, a feature known as "adaptive bitrate streaming". Video tiles are streamed ahead of time and buffered at the client before they can be rendered to the user. As the user changes their viewportthe appropriate tiles within the user's `FOV` is extracted from the client's buffer and rendered on the user's display.

Challenges of 360° video delivery. A key challenge in delivering 360° videos is that they are an order of magnitude larger in size than traditional (2D) videos [6, 7, 8]. 360° videos require multiple tiles to cover the entire viewing sphere, each encoded in multiple bitrates akin to 2D videos. Further, a high resolution of 4K to 8K is recommended for viewing AR/VR media [7]. Thus, the data rate of a 360° video that delivers a 4K stream for eight tiles and allows the user to watch the full 360° viewing sphere is 200 Mbps, compared to about 25 Mbps for a traditional 4K video. In fact, the data rate of such a 360° video is an order of magnitude larger than the US's average last-mile bandwidth [9, 10]. Additionally, when the user's viewport changes, say due to a head movement, the new tiles that fall within the user's new `FOV` must be rendered within a latency of a few tens of milliseconds so as to not cause a *rebuffering event* that results in showing either an incorrect/stale tile or no tile at all (i.e., blank screen). If the "motion-to-photon" latency exceeds a few tens of milliseconds, the user experiences a

[1]Field of view is the spatial area that falls within the viewport of the user's device. A user sees only the portion of the 360° video that is within the `FOV`.

degraded quality-of-experience, or even cybersickness [5].

Adaptive Bitrate (ABR) for 360° Videos. We investigate ABR algorithms for handling the challenges posed by the large size of 360° videos. While ABR algorithms for traditional 2D videos have been extensively studied over the past decade [11, 12, 13, 14, 15, 16, 17], ABR algorithms for 360° videos are notably more complex. They must perform both "view adaptation" by predicting the user's head position and potential future tile views, and "bitrate adaptation" by determining appropriate bitrates for downloading tiles. Importantly, these two adaptations are jointly optimized to prioritize higher bitrates for tiles more likely to be in the user's viewport. Note that this challenge is different from tile scheduling problem which determines the download ordering of tiles [18].

Challenges of Naive ABR Solutions. Naive ABR algorithms equally distribute the available bandwidth among all tiles, resulting in downloading the sametiles for each chunk. While this approach prevents rebuffering by having the entire tiles of a chunk, it leads to suboptimal video quality. An alternative approach predicts the tiles the user is likely to watch and downloads only those tiles, reducing the number of downloaded tiles and allowing for higher quality. However, this approach is susceptible to rebuffering if the user unexpectedly switches to unpredicted tiles[19, 20, 21, 22, 23, 24, 25, 26]. Our proposed approach offers a provably near-optimal solution by striking a balance between these naive extremes, achieving both high quality and reduced rebuffering.

Our Contributions. We leverage Lyapunov optimization techniques to achieve *both* high bitrates and low rebuffering by judiciously downloading higher-quality tiles for tiles that are more likely to be in the `FOV` of the users, while using lower-quality tiles for the rest of the tiles as a hedge against rebuffering. Our algorithm, `BOLA360`, is a near-optimal ABR algorithm for 360° videos that also empirically performs better than state-of-the-art algorithms. We make the following specific contributions.

1) We frame the optimization of quality-of-experience (QoE) for 360° videos as the `ABR360` problem. We model QoE as a weighted sum of two terms, one term relates to the quality (i.e., bitrate) of the video tiles viewed by the user, and the other term relates to continuous video playback without rebuffers.

2) We present an optimal offline solution to the `ABR360` problem, which establishes an upper bound on the achievable QoE by any online algorithm. While this offline algorithm is impractical for real-world use, it serves as a benchmark for comparing the QoE performance of online algorithms in our experimental analysis.

3) We present `BOLA360`[2], an algorithm that finds a near-optimal solution for `ABR360` in an online manner without the future knowledge of inputs. In each round, `BOLA360` selects a suitable bitrate for each tile based on the current buffer utilization. Further, there are multiple parameters in `BOLA360` that could be tuned to improve the performance under different conditions and environments.

4) We analyze `BOLA360`'s performance, demonstrating that (i) it never exceeds the client's buffer capacity (Theorem 5.1) and (ii) its average QoE is within a small additive constant factor of the offline optimum of `ABR360` (Theorem 5.2), the additive factor goes to zero when the buffer size goes to infinity. Additionally, considering the *playback delay* – the time between tiled download and rendering, our analysis reveals a tradeoff between playback delay and `BOLA360`'s QoE, i.e., one needs to tolerate a longer playback delay to achieve better QoE (Remark 2).

5) We implement `BOLA360` on a simulation testbed and evaluate its performance using both real and synthetic data traces. Using trace-based simulations, we compare `BOLA360` with state-of-the-art algorithms used in `VA-360` [28], `ProbDASH` [29], `Salient-VR` [30], `Flare` [31], `Pano` [32], and `Mosaic` [33]. Our results show that in comparison with QoE of the best alternative ABR algorithm, on average `BOLA360` provides 6% improvements over 14 real network profiles (Figure 9) and 9% improvements over 12 different head position probability distributions (Figure 13).

6) Finally, we explore two extensions to `BOLA360`, addressing specific real-world scenarios [34]. While `BOLA360` already demonstrates impressive QoE, average bitrate, and rebuffering performance, further enhancements can be achieved by introducing heuristics on top of its core design. We introduce

[2]A preliminary version of this article appeared at ACM MMSys 2024 [27].

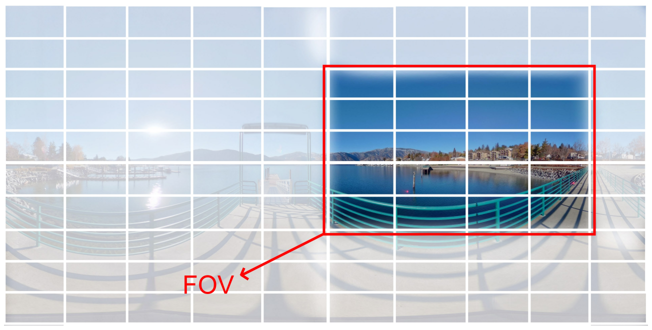


Figure 2: One shot from the entire spatial area of 360° video and FOV of user in that

BOLA360-PL and BOLA360-REP, each targeting specific limitations of the original algorithm. Our experiments show BOLA360-PL reduces reaction time by up to 67.8%, while BOLA360-REP enhances both playing bitrate and reaction time by 91.2% and 80.0%, particularly when combined with short-term head position predictions. These heuristics offer efficient and practical solutions, surpassing the original algorithm's performance.

Roadmap. The paper is structured as follows: First, we investigate the background of 360° video streaming in Section 2. Then, we present the system model and formulate the ABR360 problem in Section 3. Next, In Section 4, we present an optimal offline solution for the problem of bitrate adaptation for 360° video streaming. In Section 5, we develop BOLA360 using a Lyapunov optimization approach, proving its near-optimality. Section 6 evaluates the performance BOLA360 against state-of-the-art algorithms. In Section 7, we introduce two enhancements for BOLA360 which practically improves its performance. The related work is discussed in Section 8, and we conclude in Section 9.

2 Background

ABR Algorithm for 360° Videos. Tile-based 360° videos temporally slice the video into chunks. Each chunk is split into multiple tiles to cover the entire 360° spatial area. Usually, each tile is encoded in multiple quality levels or bitrates for video streaming. The ABR algorithm for 360° video has to select the bitrate of a tile before downloading it. So, the action of the online ABR algorithm for each chunk is a list of selected bitrates for each tile.

Field of View [FOV]. A 360° video is encoded in the full 360° visual sphere. However, the human eye's field of vision covers about 130°[35]. Therefore, the user interacting with the 360° video cannot see the entire spatial area of the presented video. The part of the 360° video inside the user's visible region is called Field of View or FOV. Figure 2 shows an example of a FOV that consists a subset of tiles of the full sphere of the 360° video seen by the user. We use the term view to refer to the group of tiles inside the FOV. When the user interacts with 360° video with a VR headset, the user can arbitrarily change the FOV and view by moving their head.

Buffer Occupancy based Lyapunov Algorithm. BOLA [34] is an ABR algorithm optimized for single-tile 2D video streaming, using buffer occupancy in bitrate selection. In 360° video streaming, besides bandwidth uncertainty, additional factors like user head direction and FOV introduce complexity. Due to these added challenges and uncertainties unique to ABR360, traditional 2D ABR algorithms cannot be directly applied to effectively solve ABR360.

3 System Model and Problem Formulation

In this section, we present the system model and problem formulation for the online 360° video bitrate selection problem.

Table 1: Summary of important notations.

Notation	Description
K	Number of chunks
D	Number of tiles
M	Number of available bitrates
δ	Length of a chunk
S_m	Size of a tile with bitrate index m
v_m	Utility value of watching a tile with bitrate index m
$p_{k,d}$	The probability of the tile d is inside FOV while playing chunk k
T_{end}	Streaming duration
T_k	Time interval between finishing downloading chunks $k-1$ and k
γ	Relative importance of the two terms in user's QoE
$Q(t)$	Buffer level at time t
$Q_{\max}$	Buffer capacity
$a_{k,d,m}$	Decision variable for bitrate index m of tile d of chunk k
n_k	Average number of tiles downloaded for chunks played during the downloading of chunk k

The 360° Video Model. We consider a 360° video as a sequence of K *chunks*, where each chunk represents δ seconds of the playback time. Each chunk is further partitioned into D *tiles* to cover the entire 360° spatial area. Each tile is encoded in M different *bitrates*, all of which are available at the server; the higher the bitrate, the larger the size in bits. Let S_m denote the size (bits) of a tile with bitrate index m. We define v_m as the utility value the user gets by watching a tile with bitrate index m. Therefore, we have the following inequality.

$$S_1 \leq S_2 \leq ... \leq S_M \Leftrightarrow v_1 \leq v_2 \leq ... \leq v_M.$$

During the playback time of each chunk, the user views only tiles inside their `FOV`. The bitrate of tiles inside the `FOV` directly impacts the QoE. Downloading tiles which falls out of `FOV` wastes the bandwidth capacity. A key challenge is that the `FOV` is unknown to the bitrate selection algorithm at download time. As a result, the online bitrate selection algorithm must predict the `FOV` and download tiles based on its prediction. Let $p_{k,d}$ denote the probability of the tile d is inside `FOV` while playing k^{th} chunk. We assume that these probability values are given from a prediction based on the previous user's watching the video [36, 19, 37, 30, 38], or from a chunk analysis of the content combined with points probability analysis of 360° sphere [29, 39, 40]. For simplicity, we assume that the probability values of tiles within a chunk are normalized, such that $\sum_{d=1}^{D} p_{k,d} = 1$.

Problem Formulation. In what follows, we formulate `ABR360`, an online optimization problem for the bitrate and view adaptation of 360° video streaming. In `ABR360`, the objective is to maximize the expected QoE of the user, including two terms: 1) the utility term that is related to quality of the video watched by the user, and 2) the smoothness of streaming term that captures continuous playback without rebuffering. The first term directly depends on the bitrate downloaded by the streaming algorithm, i.e., the higher the bitrate, the higher the utility. The second term captures the expected smoothness of video streaming. Rebuffering happens when at least one of the tiles inside `FOV` is not completely downloaded during playback time. Note that the above two terms conflict with each other. To maximize the utility, an ABR algorithm must download the highest possible bitrate tiles. However, to maximize the expected continuous smooth playback, the ABR algorithm must download low-bitrate tiles. Thus, to maximize the sum of both terms, the ABR algorithm must balance the two conflicting requirements.

We now formulate QoE mathematically to capture the utility as the sum of the two terms U_K and R_K. The first term U_K represents the time-average expected playback utility the video player prepares for the user over the sequence of chunks and is defined as

$$U_K = \frac{\sum_{k=1}^{K}\sum_{d=1}^{D}\sum_{m=1}^{M}\mathbb{E}\{a_{k,d,m}\cdot p_{k,d}\cdot v_m\}}{\mathbb{E}\{T_{\text{end}}\}}, \tag{1}$$

where T_{end} is the time the video player finishes playback of the last chunk, and $a_{k,d,m}$ is a binary optimization variable in the `ABR360` problem: $a_{k,d,m} = 1$ if bitrate index m is selected for tile d of chunk k; 0, otherwise. The second QoE term is denoted by R_K, which targets the playback smoothness as follows.

$$R_K = \frac{\sum_{k=1}^{K}\sum_{d=1}^{D}\sum_{m=1}^{M}\mathbb{E}\{a_{k,d,m}\delta\}}{\mathbb{E}\{T_{\text{end}}\}}. \tag{2}$$

That is, R_K represents the ratio of the expected playback duration of downloaded tiles to the streaming duration. A low R_k when T_{end} greatly exceeds tiles playback duration (numerator) can lead to rebuffering, making a high R_k indicative of continuous playback. Unlike U_k, R_K inversely correlates with download time (or bitrate), decreasing with higher bitrates. Expectations in Equation (1) and Equation (2) are computed over the possible randomized decisions or outcomes of the ABR algorithm solving `ABR360`.

Let t_k denotes the time the video player completes the download of tiles that belong to chunk $k-1$ and decides about the bitrate of tiles for k^{th} chunk. And T_k shows the time interval between finishing downloading chunks $k-1$ and k, i.e., $T_k = t_{k+1} - t_k$. We use the coefficient $\gamma > 0$ to set the relative importance of the two terms in the user's final QoE, i.e., γ provides an opportunity to tune the relative importance of high-bitrate streaming with respect to a continuous streaming experience. We formulate the `ABR360` problem as follows.

$$\begin{aligned}
[\texttt{ABR360}] \qquad & \max && U_K + \gamma R_K && \text{(3a)}\\
& \text{s.t.,} && \sum_{m=1}^{M} a_{k,d,m} \le 1, \quad \forall d,k, && \text{(3b)}\\
& && Q(t_k) \le Q_{\max}, \quad \forall k, && \text{(3c)}\\
& \text{vars.,} && a_{k,d,m} \in \{0,1\}. && \text{(3d)}
\end{aligned}$$

Constraint (3b) limits to select at most one bitrate for each tile of a chunk. The second constraint (3c) enforces the buffer capacity limit, where $Q(t_k)$ is the buffer level at time t_k and shows the aggregate length of tiles available in the buffer at time t_k. $Q_{\max}$ is buffer capacity and depicts the maximum aggregate length of tiles stored in the buffer. Since the number of tiles downloaded for each chunk is not fixed, the actual number of tiles that drain out from the buffer when a chunk is played can vary from chunk to chunk. To capture this, let n_k be the average number of tiles downloaded for chunks played during the downloading of chunk k. The evolution of the buffer level is characterized as

$$Q(t_{k+1}) = \max[Q(t_k) - n_k T_k, 0] + \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\delta, \tag{4}$$

where the first term refers to the length of tiles removed from the buffer during the download time of chunk k and the second term shows the length of tiles recently downloaded.

Remark 1. *For regular 2D videos with $D = 1$, number of tiles that drain out of the buffer when each chunk is played fixed, $n_k = 1$. In this particular case, $\min[Q(t_k), T_k]$ seconds drained out of the buffer after passing T_k seconds.*

4 An Optimal Offline Algorithm

In this section, we present an offline algorithm that obtains an upper bound for the optimal QoE of `ABR360`. The algorithm is listed as Algorithm 1 and is based on dynamic programming to optimally solve the `ABR360` problem, given the full knowledge of the bandwidth capacities in advance. While this algorithm is impractical since it requires the entire input a priori, we use this algorithm to evaluate the significance of the performance of the proposed online algorithm. It is worth noting that Algorithm 1 generates an upper bound for the offline solution of the `ABR360` problem formulated in Equation (3). However, it is not a fully offline algorithm since it still takes head position probability values as the input to `ABR360`. Nevertheless, the performance of Algorithm 1 is an upper bound on the performance of any online algorithm without the knowledge of future bandwidth and head movement of the user.

Now, we proceed to explain the details of Algorithm 1. First, we discretize the time into slots with length t_0. Let $r(k, t, b)$ denote the maximum possible QoE the algorithm can achieve when it downloads the first k chunks and finishes it at time t and buffer level b. The algorithm initiates the value of $r(0, 0, 0) = 0$. Then, the algorithm calculates the download time T, and rebuffering time R, of any possible action that determines the selected bitrate during downloading k^{th} chunk and evaluates the best possible performance by utilizing the values calculated for the first $(k-1)^{th}$ chunks. Let T show how long the downloading of action $\mathbf{a}$, the set of selected bitrates for each tile, would take. The algorithm may wait if the buffer is full to place the recently downloaded segments. Let b' be the state of the buffer before downloading k^{th} chunk, then the waiting time is at most $T_0 = \max[0, b' + n_k\delta - Q_{max}]$, where n_k is the number of tiles with positive bitrate for k^{th} chunk. Then, $T' = \lfloor T_0/t_0 \rfloor \times t_0$ is the conversion of T_0 to units of t_0. We round down the value of T_0/t_0 to ensure that the final calculated QoE is an upper bound for the QoE of the optimal offline solution. Also, we calculate the rebuffering happened while downloading k^{th} chunk, $R = \max[T' - b', 0]$. Finally, we calculate the impact of action $\mathbf{a}$ on achieved QoE by calculating r' in line 16. Now, by knowing the download time and rebuffering for action $\mathbf{a}$ we can update values of time, buffer, and utility achieved by the algorithm by the end of downloading k^{th} chunk as demonstrated in Lines 11, 12, and 17 of Algorithm 1. The following theorem states the optimality of Algorithm 1.

Theorem 4.1. *Algorithm 1 gives an upper bound for QoE of optimal offline solution for `ABR360`.*

Proof. The dynamic programming solution for the optimal offline problem with K chunks evaluates the QoE that can be achieved by any permutation of actions during downloading the first $K-1$ chunks. Then, it uses the QoE values to evaluate the best achievable QoE for downloading all K chunks. We prove the correctness of the algorithm by induction. The base case is evaluating the performance of the optimal offline solution for starting moment of the stream. $r(0, 0, 0) = 0$ is the base case which shows that the achieved QoE is zero at starting moment of the stream when we have downloaded no tiles and the buffer is empty. This proves the correctness of the base case. Since we are evaluating the performance of the offline algorithm for any value of t and b for $r(K-1, t, b)$, we would be able to calculate the performance of the offline algorithm for all K chunks by evaluating every possible action. It gives the optimal offline solution a chance to try any possible set of actions (line 5 of Algorithm 1) and finds the best sequence of actions that leads to maximum QoE. As a result, $r(K, t, b)$ can store the maximum achieved QoE by downloading K chunks until time t and using b seconds buffer. As a result, after filling the dynamic programming table, the value of $r(K, t, b)/(t + b)$ shows the upper bound on QoE of the optimal offline algorithm, and the proof ends.

Algorithm 1: Optimal offline algorithm for `ABR360`

```
Result: $\max_{(t,b)} r(K,t,b)/(t+b)$
$r(k,t,b) = -\infty$;
$r(0,0,0) = 0$;
for $k$ in $[1,2,...,K]$ do
    for all $(t',b')$ such that $r(k-1,t',b') > -\infty$ do
        for all possible set of action $\mathbf{a} = [m^{(1)}, m^{(2)}, ..., m^{(D)}]$ do
            $n$ : number of positive bitrates in $\mathbf{a}$;
            $T$ : download-time($\mathbf{a}$);
            $T_0 = \max[T, b' + n\delta - Q_{max}]$;
            $T' = \lfloor T_0/t_0 \rfloor \times t_0$;
            $R = \max[T' - b', 0]$;
            $t = t' + T'$;
            $b = b' - T' + R + n\delta$;
            $\forall d,\ v^{(d)}$ := utility value of tile $d$;
            $\forall d,\ a_{(d)}$ := 1 if $m^{(d)} > 0$ else 0;
            $r_k = \sum_{d=1}^{D} a_{(d)}(v^{(d)} p_{k,d} + \gamma\delta)$;
            $r' = r(k-1,t',b') + r_k$;
            $r(k,t,b) = \max[r(k,t,b), r']$;
        end
    end
end
```

5 `BOLA360`: An Online 360° ABR Algorithm

In this section, we propose `BOLA360`, a Lyapunov-based algorithm that finds a near-optimal solution to `ABR360`. `BOLA360` is an online algorithm whose decisions do not require the knowledge of future bandwidth values.

5.1 Design and Analysis of `BOLA360`

The design of `BOLA360` is based on three key ideas. First, `BOLA360` finds a solution for a single-slot maximization problem that leads to a near-optimal solution for the original long-term problem over K chunks. Note that, solving the long-term optimization problem is not possible for the online algorithm since there is uncertainty about the future input. Second, the single-slot decision of `BOLA360` is based on the buffer level; the higher the current buffer level, the higher the selected bitrate for download. This is intuitive since a high buffer level indicates that the input rate into the buffer was higher than the output rate from the buffer, so the algorithm has more freedom to download high-quality tiles. Third, `BOLA360` uses a threshold as the indicator of high buffer utilization, and by reaching the threshold, it moves to an idle state and waits until the buffer level decreases again. This approach limits the buffer utilization of `BOLA360`. It is worth noting that at the beginning and with an empty buffer, `BOLA360` starts downloading low bitrates. With the above three key ideas, we now explain the technical details of `BOLA360`. The pseudocode for action taken by `BOLA360` for chunk k is described in Algorithm 2.

`BOLA360` uses an input parameter V that controls the trade-off between the performance of the algorithm and the maximum acceptable buffer utilization of the algorithm. Note that parameter V also plays a critical role in the playback delay, i.e., for real-time streaming, smaller values of V are preferable, while in an on-demand streaming application, the larger values of V are acceptable. At the decision time t_k for chunk k, the buffer level $Q(t_k)$ and head position probability values encoded in

$p_{k,d}$ are given. BOLA360 selects the bitrates for tiles of chunk k by solving the maximization problem described in the following.

$$\underset{a(k)}{\arg\max} \quad \eta(k, a(k)) = \sum_{d=1}^{D} \sum_{m=1}^{M} \frac{a_{k,d,m}(V(v_m \ . \ p_{k,d} + \gamma\delta) - Q(t_k)/\delta \ )}{S_m} \tag{5a}$$

$$\text{s.t.,} \qquad \sum_{m=1}^{M} a_{k,d,m} \leq 1, \quad \forall k, d, \tag{5b}$$

$$\text{vars.,} \qquad a_{k,d,m} \in \{0, 1\}, \tag{5c}$$

where

$$a(k) := \{a_{k,d,m} | \forall k, m\},$$

is a decision vector of BOLA360 and

$$0 < V \leq \frac{Q_{\max}/\delta - D}{v_M + \gamma\delta},$$

is a control parameter bounded by the R.H.S term to guarantee that the required buffer level for BOLA360 is less than $Q_{\max}$. Constraint (5b) limits BOLA360 to select at most one bitrate for each tile. BOLA360 selects the near-optimal bitrates of chunk k by finding a decision vector $\mathbf{a}(k) = [a_{k,1,1}, a_{k,1,2}, ..., a_{k,1,M}, a_{k,2,1},$ that maximizes the value of $\eta(k, a(k))$ in Equation (5a). When the buffer level exceeds $V\delta(v_M + \gamma\delta)$, the algorithm enters the idle state and downloads nothing. In this situation, BOLA360 waits for Δ seconds and repeats the bitrate selection for that chunk again. The selection of Δ could be dynamic as suggested in [16], the algorithm waits until the buffer level reaches $Q(t_0) \leq V\delta(v_M + \gamma\delta)$. We note that our theoretical analysis is valid even with a dynamic waiting time.

Algorithm 2: BOLA360 (k)

```
1 a(k): A decision vector that maximizes the value of $\eta(k, a(k))$ defined in (5a) with respect to
    single-bitrate constraint (5b) for chunk $k$;
2 if number of non-zero elements in a(k) > 0 then
3   | Download bitrates according to a(k) and finish the decision making of chunk $k$;
4 end
5 else
6   | Wait for $\Delta$ seconds and repeat the bitrate selection for this chunk again;
7 end
```

5.2 Theoretical Analysis of BOLA360

We first provide an upper bound for the buffer level of BOLA360 in Theorem 5.1. Second, in Theorem 5.2, we show the QoE of BOLA360 is within a constant term of the optimal QoE of ABR360. The theoretical results reveal an interesting trade-off between the QoE and the playback delay of the BOLA360, which is discussed in Remark 2.

Theorem 5.1. *Under bitrate control of BOLA360, the buffer level never exceeds $V\delta(v_M + \gamma\delta) + D\delta$.*

Proof. The proof of this theorem is inspired by the proof of Theorem 1 in [16]. However, for BOLA360, one has to deal with another challenge originated by adding head position probabilities into the control plane of BOLA360. The high-level idea is BOLA360 select bitrates if the buffer level is at most $V\delta(v_M + \gamma\delta)$, otherwise it enters the idle states. Therefore, the maximum possible value of buffer level after download of new tiles would be $V\delta(v_M + \gamma\delta) + D\delta$. We prove this theorem by induction. The base case is $Q(t_1) = 0 \le V\delta(v_M + \gamma\delta) + D\delta$ that satisfies the statement of the theorem. Two cases are possible for value of $Q(t_k)$:

- *Case 1*: $Q(t_k) \le V\delta(v_M + \gamma\delta)$: in this case, the buffer level at time t_{k+1} will not exceed $Q(t_k) + D\delta \le V\delta(v_M + \gamma\delta) + D\delta$.
- *Case 2*: $V\delta(v_M + \gamma\delta) < Q(t_k) \le V\delta(v_M + \gamma\delta) + D\delta$: In this scenario, the action at time t_k is to wait and refrain from downloading any tiles, as downloading any bitrate m would introduce a negative term into the value of $\eta(k, \mathbf{a}(k))$ in Equation (5a). Thus, $Q(t_{k+1}) \le Q(t_k)$.

Now, we proceed to analyze the QoE of `BOLA360`. With large K, the `ABR360` problem with rate stability constraint [41] is equivalent to the relaxed version of `ABR360` with limited buffer capacity, i.e.,

$$\begin{aligned} Q(t_k) \le & Q_{\max} \\ & \Rightarrow \lim_{K\to\infty} \frac{1}{K}\mathbb{E}\Big\{\sum_{k=1}^{K}\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\delta\Big\} \le \lim_{K\to\infty}\frac{1}{K}\mathbb{E}\Big\{\sum_{k=1}^{K} n_k T_k\Big\}. \end{aligned}$$

In addressing the `ABR360` problem with limited buffer capacity, it's essential to ensure that the expected input rate into the buffer remains below the buffer's output rate. Failing to do so could lead to a buffer capacity breach, especially as K approaches infinity. Notably, solutions that accommodate limited buffer capacity inherently satisfy the rate stability constraint, though the reverse may not always hold. In addition, in the limited buffer capacity setting, the difference between $\mathbb{E}\{T_{\text{end}}\}$ and $\mathbb{E}\{\sum_{k=1}^{K} T_k\}$ is bounded by a finite value of $Q_{\max}$. Consequently, for the large videos, Equations (1) and (2) allow the substitution of $\mathbb{E}\{T_{\text{end}}\}$ with $\mathbb{E}\{\sum_{k=1}^{K} T_k\}$.

The stationary algorithm. In the context of `ABR360` problem, we define *stationary algorithm* as an ABR algorithm that uses a fixed set of bitrates, $\mathbf{A}^*$, with size D ($|\mathbf{A}^*| = D$), and for each chunk k, the set of selected bitrates for all D tiles are the same as the $\mathbf{A}^*$. Note that the selected bitrate for each tile may vary over time depending on the head position probability values, while the set of bitrates selected for all tiles of the chunk remains fixed.

Offline `ABR360` problem fits in the notation of optimization for renewal frames [42]. Precisely, by setting renewal frame duration the same as chunk download times and letting the achieved QoE of downloading each chunk represent penalty values in the notation of [42], the offline `ABR360` problem can convert into an optimization problem over renewal frames. Then, following Lemma 1 in [42], we prove the existence of a stationary algorithm with optimal QoE of $U_K^* + \gamma R_K^*$.

Lemma 1. *For the `ABR360` with a large video, i.e., $K \to \infty$, there exists a stationary algorithm that satisfies the rate stability constraint and achieves the optimal expected QoE of $U_K^* + \gamma R_K^*$.*

Proof Sketch. The proof of this lemma follows from Lemma 1 in [42] and continues with the approach taken for proof of Lemma 1 in [16]. Based on the definition of a stationary algorithm for the `ABR360` problem, the expected QoE of the stationary algorithm is the same as expected, achieving QoE on each slot, which satisfies the criteria of Lemma 1 in [42].

Theorem 5.2 (main theorem)**.** *Let `OBJ` be the expected QoE achieved by `BOLA360`. For a large video, i.e., $K \to \infty$,*

$$\texttt{OBJ}^* - \frac{D\delta^2 + \Psi}{2V\delta^2}\sigma \le \texttt{OBJ}, \tag{7}$$

where $\texttt{OBJ}^* = U_K^* + \gamma R_K^*$ is expected QoE of the offline optimal algorithm, and $\sigma = 1/\mathbb{E}\{T_k\}$ and $\Psi \le \mathbb{E}\{DT_k^2\}$. That is, `BOLA360` achieves a QoE that is within an additive factor of the offline optimal.

Proof. Let's define the Lyapunov function $L(Q(t_k))$, and per-slot conditional Lyapunov drift $\Phi(t_k)$ as below

$$\begin{aligned}L(Q(t_k)) &= \frac{1}{2\delta^2}Q^2(t_k),\\ \Phi(t_k) &= \mathbb{E}\{\Delta\ L(Q(t_k)) \mid Q(t_k)\} = \mathbb{E}\{L(Q(t_{k+1})) - L(Q(t_k))|Q(t_k)\}.\end{aligned}$$

Consider two cases: 1) $Q(t_k) \le n_kT_k$ and, 2) $Q(t_k) > n_kT_k$. The value of $\Phi(t_k)$ can be derived from the buffer level evolution described in Equation (4).In the first case, when $Q(t_k) \le n_kT_k$, the buffer is emptied after downloading chunk k, and the value of $\Phi(t_k)$ is given by:

$$\Phi_1(t_k) = \mathbb{E}\{\frac{1}{2}(\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m})^2 - \frac{1}{2\delta^2}Q^2(t_k)|Q(t_k)\},$$

In the second case, where $Q(t_k) > n_kT_k$, we have:

$$\Phi_2(t_k) = \mathbb{E}\{\frac{1}{2}(\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m} - \frac{n_kT_k}{\delta})^2 - \frac{Q(t_k)}{\delta}(\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m})\}|Q(t_k)\Big\}.$$

Thus, the value of $\Phi(t_k)$ can be expressed as:

$$\begin{aligned}\Phi(t_k) &\le \max\{\Phi_1(t_k), \Phi_2(t_k)\}\\ &\le \mathbb{E}\Big\{\frac{n_k^2T_k^2 + \delta^2(\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m})^2}{2\delta^2}|Q(t_k)\Big\} - Q(t_k)\mathbb{E}\Big\{\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\}|Q(t_k)\Big\}\\ &\le \mathbb{E}\Big\{\frac{D\delta^2 + n_k^2T_k^2}{2\delta^2}|Q(t_k)\Big\} - Q(t_k)\mathbb{E}\Big\{\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\}|Q(t_k)\Big\}\\ &= \frac{D\delta^2 + \Psi}{2\delta^2} - Q(t_k)\mathbb{E}\Big\{\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\}|Q(t_k)\Big\}.\end{aligned}$$

By subtracting a fixed term from both sides we get:

$$\begin{aligned}\Rightarrow \Phi(t_k) - V\mathbb{E}\Big\{\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\Big\}\\ \le \frac{D\delta^2 + \Psi}{2\delta^2} - \frac{Q(t_k)}{\delta}\mathbb{E}\Big\{\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}|Q(t_k)\Big\}\\ -V\mathbb{E}\Big\{\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\Big\}\\ \le \frac{D\delta^2 + \Psi}{2\delta^2} - \frac{Q(t_k)}{\delta}\mathbb{E}\Big\{\frac{n_kT_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}|Q(t_k)\Big\}\\ -V(U_K^* + \gamma R_K^*)\mathbb{E}\{T_k|Q(t_k)\}.\end{aligned}$$

The previous equation holds since the decision of `BOLA360` at time t_k is a solution of the maximization equation detailed in Equation (5) and

$$\mathbb{E}\{\sum_{d=1}^{D}\sum_{m=1}^{M} a^*_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\} \le \mathbb{E}\{\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\}.$$

Then, we have

$$\begin{aligned}
\Phi(t_k) - V\mathbb{E}\Big\{\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\Big\} \\
\leq \frac{D\delta^2 + \Psi}{2\delta^2} - \frac{Q(t_k)}{\delta}\Big(\frac{\mathbb{E}\{n_k\}}{\delta} - \frac{\mathbb{E}\{\sum_{d=1}^{D}\sum_{m=1}^{M} a^*_{k,d,m}\}}{\mathbb{E}\{T^*_k\}}\Big)\mathbb{E}\{T_k\} \\
-V(U^*_K + \gamma R^*_K)\mathbb{E}\{T_k\},
\end{aligned}$$

where $a^*_{k,d,m}$ is the action of stationary algorithm for tile d and bitrate index m of chunk k which satisfies the rate stability constraint. T^*_k shows the length of download time for chunk k while the stationary algorithm is taking action. Based on rate stability constraint, the second term in the equation above is always negative,

$$\begin{aligned}
\sum_{k=1}^{K}\Phi(t_k) - \sum_{k=1}^{K} V\mathbb{E}\Big\{\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}(p_{k,d}v_m + \gamma\delta)|Q(t_k)\Big\} \\
\leq \frac{D\delta^2 + \Psi}{2\delta^2}K - V(U^*_K + \gamma R^*_K)\mathbb{E}\{T_k\}K.
\end{aligned}$$

By dividing all terms by $V \cdot K \cdot \mathbb{E}\{T_k\}$ and taking the limit $K \to +\infty$, the proof is completed, noting that the total download time is at most $Q_{\max}$ seconds shorter than T_{end}.

Remark 2 (On the conflict between the playback delay and QoE of streaming)**.** *Theorem 5.2 states as the value of V increases, the performance of BOLA360 gets closer to the optimal QoE. However, Theorem 5.1 reveals that the upper bound on the playback delay increases with higher values of V. Comparing these results, we observe a trade-off between minimizing playback delay and maximizing QoE in BOLA360. As the playback delay increases, the QoE performance of BOLA360 approaches the offline optimum.*

5.3 Understanding the Behavior of BOLA360

We demonstrate the functionality of BOLA360 through a straightforward test. Our test utilizes a 250-second video, segmented into 5-second chunks. Each chunk further divides into six tiles, each encoded at distinct bitrates: 2Mbps, 4Mbps, 6Mbps, 8Mbps, 10Mbps, and 15Mbps. To represent utility values, we employed a logarithmic function $v_m = \log(2S_m/S_1)$, similar to previous works such as [43, 16, 44]. While our theoretical results only require a non-decreasing utility function, we opted for a concave function that better reflects real-world utility functions. The concave utility function exhibits a diminishing return property, meaning that increasing the bitrate from 1 Mbps to 2 Mbps provides more utility than increasing it from 10 Mbps to 11 Mbps, even though the bitrate difference is the same in both cases. In Table 2, we report the utility values generated from logarithmic function, size of tiles, and available bitrates. For this simple test, we set $\gamma = 0.1$ and $V = 5.5$. We note that U_K assesses the expected utility across various tiles within a chunk, while R_K quantifies the aggregate length of downloaded tiles. Setting $\gamma = 1/D$ equalizes the significance of utility and smoothness concerning a single tile.

The head position of the user is represented by a probability distribution that is critical for guiding the actions of BOLA360. For this test, we evaluate the performance of BOLA360 using two different head position probability distributions. The first distribution is homogeneous, where each tile is assigned a uniform probability, resulting in an equal likelihood of the user watching any tile ($p_{k,d} = 1/D$ for all tiles). The second distribution is heterogeneous, with a linear increase in probability from the minimum to the maximum. Specifically, we set the maximum and minimum probabilities as 0.317 and 0.017,

Table 2: Available bitrates and utility values used in Section 5.3

Bitrate (Mbps)	2	4	6	8	10	15
Sizes (Mb)	10	20	30	40	50	75
Utility values	1.000	2.000	2.585	3.000	3.322	3.907

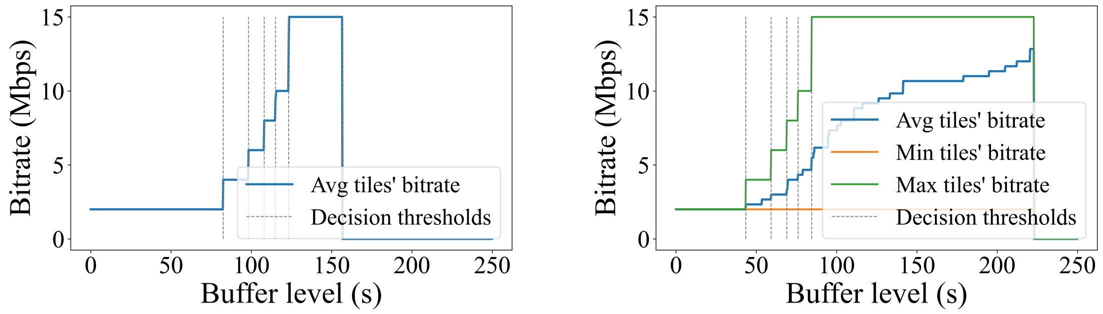


Figure 3: The selected bitrate of `BOLA360` for tiles with highest and lowest probability and average selected bitrate as a function of buffer level for homogeneous (left) and heterogeneous (right) distributions.

Table 3: Two probability distributions used in Section 5.3

Distribution	Probabilities of head direction in an descending order					
Homogeneous	0.166	0.166	0.166	0.166	0.166	0.166
Heterogeneous	0.317	0.257	0.197	0.136	0.076	0.017

respectively. The values of $p_{k,d}$ for these two head position probability distributions are listed in Table 3. Note that these values are chosen arbitrarily to elucidate `BOLA360`'s behavior clearly.

Figure 3 shows the maximum, minimum, and average bitrates of downloaded tiles for each chunk of the video. For the homogeneous distribution, the selected bitrate for all tiles of a chunk is the same. The results in Figure 4 show that the average download bitrate grows with an increase in buffer level. We show the threshold values for the buffer level where the action for the tile with the highest probability changes. In addition, we show the variations of buffer level over time for both homogeneous and heterogeneous head position probability distributions in Figure 4. When the buffer level is higher

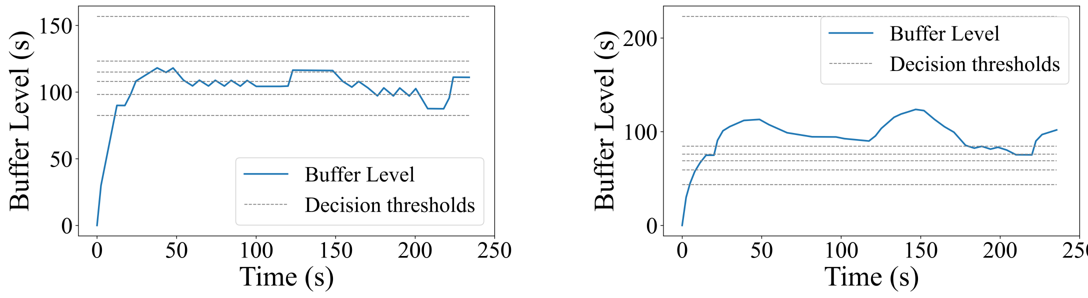


Figure 4: Buffer level variation over time under bitrate selection of `BOLA360` for homogeneous (left) and heterogeneous (right) distributions.

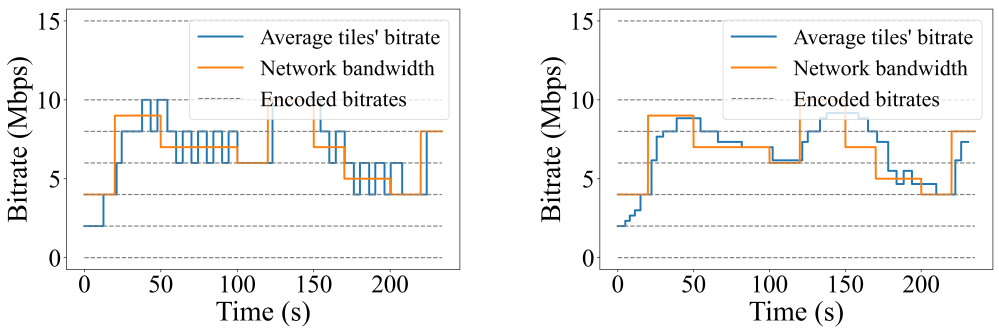


Figure 5: Variation of average downloaded bitrates over time under bitrate selection of `BOLA360` for the homogeneous (left), and heterogeneous (right) head position probability distribution.

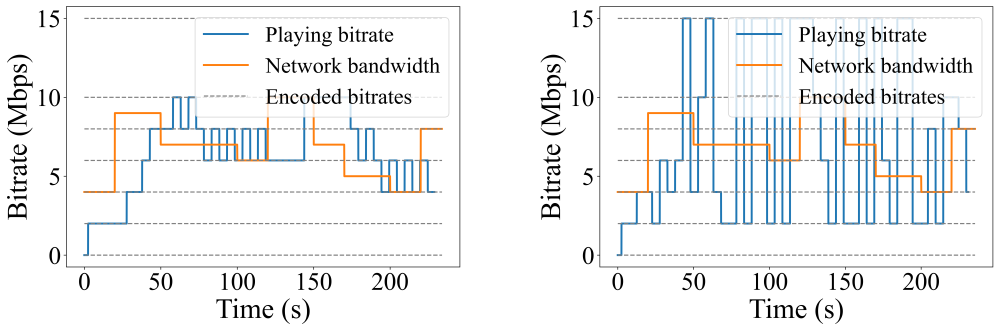


Figure 6: Variation of playing bitrate over time under bitrate selection of `BOLA360` for the homogeneous (left), and heterogeneous (right) head position probability distribution.

than $V\delta(v_M \cdot p_{k,d} + \gamma\delta)$, `BOLA360` downloads nothing for that tile. Note that increasing the value of γ increases the importance of continuous playback. Increasing the value of γ by ϵ is similar to reducing the buffer level by $\epsilon\delta^2 V$, resulting in `BOLA360` using correspondingly higher threshold values for the buffer levels for bitrate switches. Therefore, increasing the value of γ shifts the bitrate curves in Figure 3 to the right and vice versa. Lastly, Figure 5 and Figure 6 show the average bitrates of tiles downloaded across the time and the bitrate of the tiles that user actually sees (playing bitrate) in their `FOV`. One can see that `BOLA360` responds to the bandwidth change by increasing/decreasing selected bitrates.

6 Comparison Algorithms

We compare `BOLA360` with `VA-360` [28], `ProbDASH` [29], `Flare` [31], `Salient-VR` [30], `Pano` [32], and `Mosaic` [33], the leading ABR algorithms for `ABR360`. Our analysis showcases the advancements our approach brings over the state-of-the-art. While some of these algorithms like `Pano`, `Flare`, and `ProbDASH` consider additional factors such as minimizing bitrate variance among tiles within a chunk, they rely on an MPC algorithm [14] to select the aggregate bitrate for each chunk. This method's reliance on estimated bandwidth throughput poses challenges, potentially hindering their ability to achieve near-optimal QoE, a limitation shared by algorithms like `VA-360` and `Mosaic`. We used the suggested hyper-parameters from each algorithm's respective literature.

Table 4: Available bitrates and utility values of them for experiments of Sections 6.2, and 6.3

bitrate (Mbps)	0.44	0.7	1.35	2.14	4.1	8.2	16.5
Sizes (Mb)	0.88	1.4	2.7	4.28	8.2	16.4	33.0
Utility values	1.000	1.667	2.617	3.282	4.220	5.220	6.229

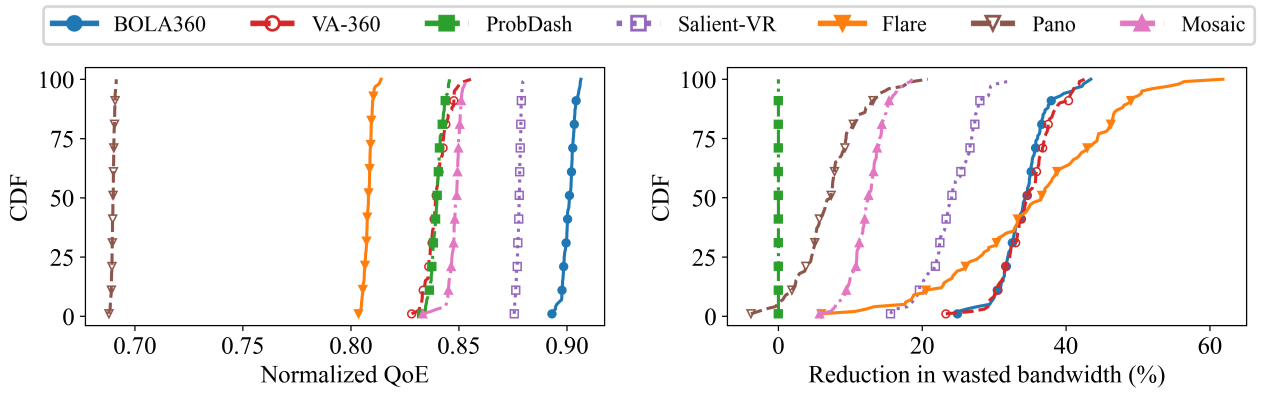


Figure 7: The CDF of normalized QoE (left) and reduction in wasted bandwidth compared with the ProbDASH (right) for BOLA360 and comparison algorithms using real network and head movement traces. The QoE of BOLA360 was higher than the QoE of other algorithms in all test trials.

6.1 Experimental Setup

We conduct multiple experiments to demonstrate the algorithms' performance under different settings. We use a 500-second video, split into chunks of 2 seconds and 8 tiles. Also, each tile is encoded in seven different bitrates - 440Kbps, 700Kbps, 1.35Mbps, 2.14Mbps, 4.1Mbps, 8.2Mbps, and 16.5Mbps. Similar to Section 5.3, we use a logarithmic utility function. The list of available bitrates, size of segments, and utility values are listed in Table 4. Although BOLA360 performs better using larger buffers, we limit the buffer capacity to $Q_{\max} = 128\delta$, which is equivalent to 32 seconds of 360° playback time, that falls within the range of suggested buffer capacity for VOD streaming [45, 46] to ensure fairness. We employ dynamic value selection for Δ, as proposed in [16], and empirically determine $V = 24.0$. Also, we set $\gamma = 0.2$ using the parameter selection methodology for γ and V outlined in Section VI of [16]. Consequently, in this scenario, the smoothness term of QoE is equivalent to the utility derived from downloading tiles at a bitrate of 2.1Mbps (equivalent to 480p resolution). We use 4G bandwidth traces from [47] and 4G/LTE bandwidth trace dataset [48] collected by IDLAB [49] to simulate the network condition. We select 14 different traces (network trace index 1 to 14 of the dataset) from 4G/LTE dataset to evaluate the performance of BOLA360 under different network conditions. The video is stored on an Apache server. Both server and client use Microsoft Windows, 24GB of RAM, and an 8-core 3Ghz Intel Core-i7 CPU. We use Chrome DevTools API [50] to transfer the video between server and client and emulate the network condition. We fetch the bandwidth capacity from the 4G/LTE dataset and inject it into the Chrome DevTools to limit the download capacity between the server and the client. In our experiments, FOV includes a single tile and unless otherwise mentioned, to capture the actual FOV of the head position probability values, we generate the navigation graph [38] for 360° video using public VR head traces [51].

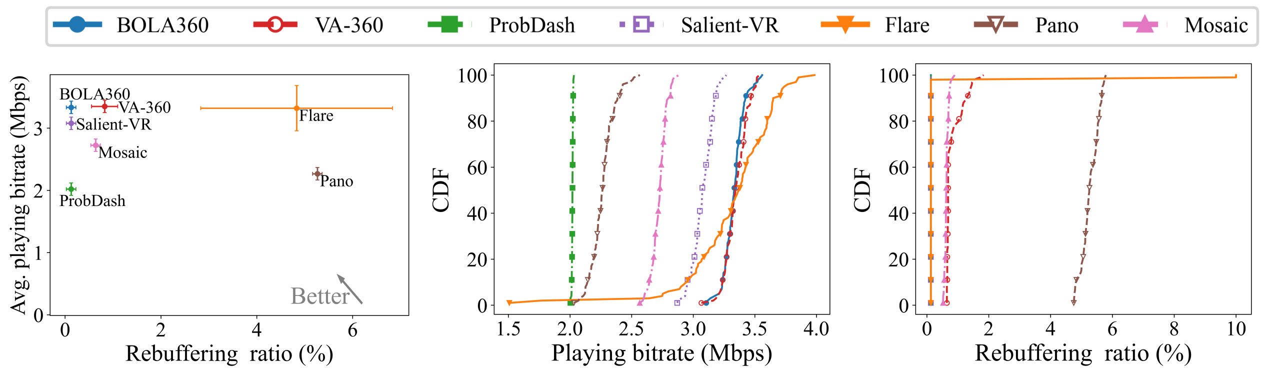


Figure 8: Average playing bitrate vs. rebuffering ratio (left), the CDF of playing bitrate (midle), and rebuffering ratio (right) of `BOLA360` and comparison algorithms using real network and head movement traces. The average playing bitrate of `BOLA360` is $3.34 Mbps$, while this value for `Salient-VR`, and `Mosaic` and `VA-360` are $3.07 Mbps$, $2.72 Mbps$, and $3.35 Mbps$. The average rebuffering for `BOLA360`, `Salient-VR`, `Mosaic`, and `VA-360` were 0.12%, 0.13%, 0.63% and 0.83%.

6.2 Performance Evaluation using Real Network and Head Movement Traces

First, we compare the performance of `BOLA360` with others using real network and head movement traces. We use a representative real 4G bandwidth trace from [47] for this comparison. We report playing bitrate, the rebuffering ratio (percentage of length of video considered as a rebuffering), and normalized QoE (QoE divided by the QoE of optimal offline algorithm) of `BOLA360`, and state-of-the-art algorithms. Additionally, we report the reduction in wasted bandwidth—the fraction of bandwidth used to download unseen tiles—for all algorithms, relative to `ProbDASH`, which exhibited the highest wasted bandwidth among the compared algorithms. Note that the average playing bitrate reported in Figure 8 is calculated over the tiles the user has seen inside `FOV`. We report the results of 100 different trials, where for each trial, we sample the user's head direction from the head position probability distribution and use the same network traces and algorithm parameters. The CDF plot of average playing bitrates, rebuffering ratio, and normalized QoE values of 100 different trials is reported in Figure 7 and Figure 8.

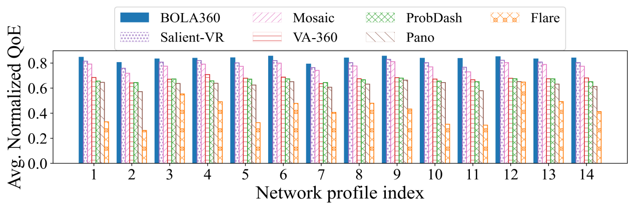
The results in Figures 7 and 8 show that `BOLA360` outperforms other comparison algorithms in QoE, and its playing bitrate was slightly less than the playing bitrate of `VA-360`, which prepares the highest playing bitrates among comparison algorithms. `VA-360` selects relatively high bitrates for all tiles of a chunk while `BOLA360` efficiently distributes the available bitrates among different tiles such that `BOLA360` can achieve a lower rebuffering ratio. Additionally, the reduction in wasted bandwidth compared to `ProbDASH` is illustrated in Figure 7. On average, `BOLA360` achieves a 34.3% reduction in wasted bandwidth compared to `ProbDASH`.

Key takeaway. `BOLA360` outperforms comparison algorithms in terms of QoE as it is designed to maximize it. Besides, no algorithm outperforms `BOLA360` on both playing bitrate and rebuffering ratio at the same time.

6.3 Impact of Network Bandwidth

In this experiment, we investigate the impact of different network profiles on the performance of ABR algorithms. We use network traces index 1 to 14 from the 4G/LTE dataset [48] to generate the bandwidth throughput. We use the same video and algorithm/problem parameters (details in Section 6.1) for all algorithms to capture the impact of the network capacity on their performance.

Figure 9 shows the average normalized QoE, playback delay, rebuffering ratio, and average playing bitrate of `BOLA360` and five comparison algorithms over 100 trials for 14 network profiles. `BOLA360` stands as the best algorithm in all 14 experiments. In this experiment, `VA-360` selects relatively higher



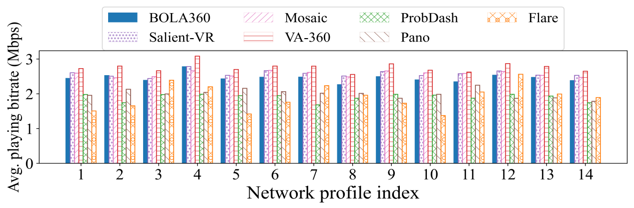
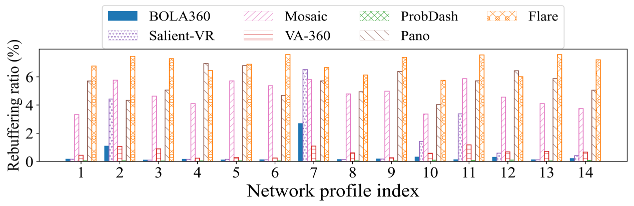


Figure 9: The average normalized QoE (left) and average playing bitrate (right) over the bitrate selection of `BOLA360` and other comparison algorithms for 14 different network profiles and 100 trials. In terms of QoE, `BOLA360` outperforms others in all profiles. On average, `BOLA360` provides about 6% improvement to the QoE of `Salient-VR`, and 110% to QoE of `Flare`.



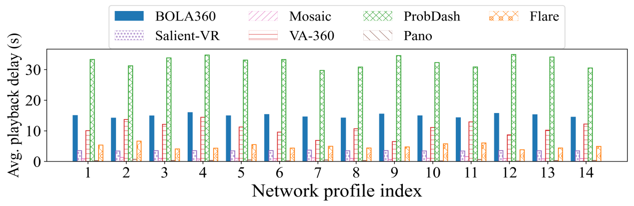


Figure 10: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of `BOLA360` and comparison algorithms for 14 different network profiles and 100 trials. `Pano` and `Flare` usually show higher rebuffering than the other algorithms, while their playback delay is shorter. The average playback delay for `BOLA360` is 14.9 seconds.

bitrates compared to other algorithms, while its high rebuffering, shown in Figure 10, lowers the QoE of this algorithm. In addition, the playback delay of `BOLA360` and comparison algorithms are shown in Figure 10. The playback delay of `VA-360` was the lowest in all experiments. That clearly shows the trade-off between having low rebuffering or low playback delay. The results show that `BOLA360` keeps the playback delay under 14.9 seconds with an average rebuffering ratio of less than 0.4%. This playback delay is consisent with the result of Theorem 5.1 and the fact that `BOLA360` tries to keep the buffer level high. Additionally, the reduction in wasted bandwidth, as compared to `ProbDASH`, for `BOLA360` and other comparison algorithms over 100 trials across 14 network profiles is illustrated in Figure 11. `Flare` achieves the highest reduction in wasted bandwidth, decreasing it by an average of 58.5%. For `BOLA360` and `Pano`, the reductions were 44.9% and 32.5%, respectively.

Key takeaway. Networks with high fluctuations (e.g., profile indexes 2 and 7) cause a higher rebuffering ratio; nevertheless, `BOLA360` keeps QoE and playing bitrate relatively high in all profiles and outperforms all alternatives.

6.4 Impact of Head Position Probabilities

The head position probability values directly impact the QoE characterized in Equations (1) and (2); hence, the performance of algorithms varies depending on these probabilities. To observe the impact of head position probabilities on the performance of ABR algorithms, we define 12 probability distributions and evaluate the performance of `BOLA360` and other algorithms against them while the rest of the setting is similar to the experiment in Section 6.2. Specifically, for each chunk k, we replace the set of probabilities with the probabilities calculated from Equation (12). Each head position probability distribution could be interpreted as a different 360° video file.

We generate the head position probability distributions based on three parameters $D_{\text{pos}}(k)$, $r(k)$, and $\alpha_p(k)$. Parameter $D_{\text{pos}}(k)$ shows the number of tiles that there is a chance to be inside `FOV` for chunk k; $r(k)$ represents the ratio between the minimum and maximum probabilities among probabilities of

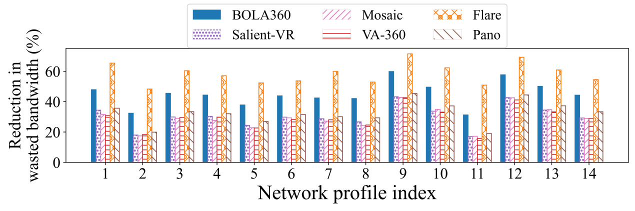


Figure 11: Reduction in wasted bandwidth compared to ProbDASH over the bitrate selection of BOLA360 and comparison algorithms using 14 different network distributions over 100 trials. The reduction of the wasted bandwidth under bitrate control of BOLA360 was between 31.3% and 59.8% over different network profiles.

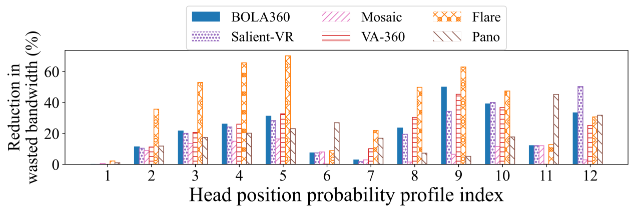


Figure 12: Reduction in wasted bandwidth compared to ProbDASH over the bitrate selection of BOLA360 and comparison algorithms using 12 different head position probability distributions over 100 trials. The reduction of the wasted bandwidth under bitrate control of BOLA360 was between 1.2% and 49.9%.

Table 5: The details of the probability distributions used in the experiment of Section 6.4

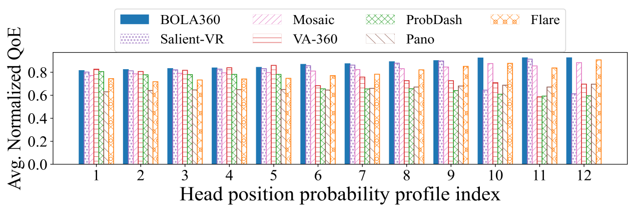
Probability profile index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$D_{\text{pos}}(k)$	8	8	8	8	8	4	4	4	4	4	2	2
$\alpha_p(k)$	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1	0	0.5

tiles for chunk k. Last, parameter $\alpha_p(k)$ determines the heterogeneity of the head position probability values for chunk k. We define the probability of i^{th} highest probable tile as a function of $\alpha_p(k)$ as follows.

$$p_i(k) = \frac{1-\alpha_p(k)}{D_{\text{pos}}(k)} + \alpha_p(k) p_i^{(L)}\Big(k, r(k)\Big), \tag{12}$$

where $p_i^{(L)}(k, r(k))$ shows the probability of i^{th} highest probable tile assuming a fixed step between probabilities in ascending order. With the above definition in Equation (12), $\alpha_p(k)$ determines the range of probabilities where $\alpha_p(k) = 0$ signifies uniform tile probabilities, while $\alpha_p(k) = 1$ indicates a wider probability range, reflecting diverse head position probability values. A justification for this model as a representative of real-world head direction prediction is that $(D - D_{\text{pos}})$ shows the number of tiles the FOV prediction model is confident that they will be out of FOV. On the other hand, $\alpha_p(k)$ shows how concentrated the FOV prediction model is. We use $r(k) = 0.05$ for all distributions. Although it's impractical to cover every possible distribution, our selection involves a low value for $r(k)$, and wide range of values for D_{pos} and $\alpha_p(k)$ to achieve broader representation. Details of the 12 probability distributions used in this section are outlined in Table 5.

We report the average normalized QoE, playback delay, rebuffering ratio, and average playing bitrate of 100 trials of BOLA360 and comparison algorithms using each head position probability distribution profile in Figures 13 (average normalized QoE and playing bitrate), and 14 (average rebuffering ratio and playback delay). Figure 13 shows that BOLA360 achieves slightly higher QoE when the prediction of FOV is concentrated on fewer number of tiles. A notable observation demonstrates that BOLA360 kept the QoE at a high value for every probability profile, while the achieved playing bitrate is promising, and kept rebuffering ratio close to the lowest among all algorithms. Finally, Figure 12 presents the reduction in wasted bandwidth relative to ProbDASH for BOLA360 and other comparison algorithms, based on 100 trials across 12 different head position probability distributions. The reduction in wasted bandwidth achieved by BOLA360 ranged from 1.2% to 49.9%, depending on the accuracy of FOV prediction across these head position probability profiles.



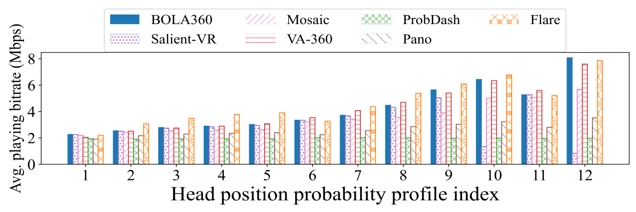
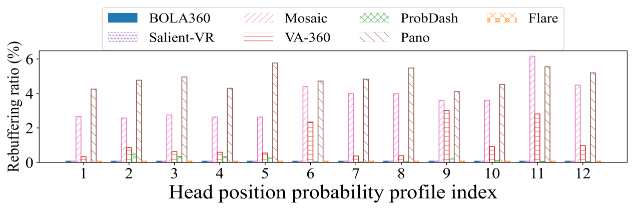


Figure 13: The average normalized QoE (left) and average playing bitrate (right) over the bitrate selection of BOLA360 and comparison algorithms using 12 different head position probability distributions over 100 trials. On average, BOLA360 provides 9% improvement to the QoE of Salient-VR, and 31% to QoE of Pano.



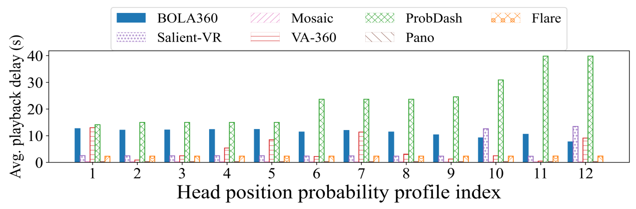


Figure 14: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of BOLA360 and comparison algorithms using 12 different head position probability distributions over 100 trials. VA-360 usually results in a high rebuffering ratio and short playback delay. Meanwhile, the rebuffering ratio of BOLA360 is slightly more than the lowest in all experiments. The average playback delay for BOLA360 is 11.1 seconds.

Key takeway. The playing bitrate of BOLA360 and most comparison algorithms improves when the head position prediction is concentrated on fewer tiles. Meanwhile, BOLA360 improves the playing bitrate more than other comparison algorithms as the head position values concentrate on fewer tiles.

6.5 Discussion on the Performance of Predictions-based Baseline Algorithms

This section provides details on why the baseline or state-of-the-art algorithm used in Section 6.1 may fail to perform well in particular scenarios, and they cannot guarantee their performance in the worst-case scenario. All of VA-360, ProbDASH, Flare, Pano, and Salient-VR algorithms take action based on the prediction of bandwidth that is given to them. The accuracy of this prediction significantly impacts the performance of these algorithms such that a prediction with an error may result in a significant difference between the performance of the ABR algorithm and the performance of the optimal offline solution. In addition, these algorithms behave similarly to the ABR360 with different values of γ. For example, for tiny values of γ, the bitrate level of the segments are much more important to the user than the smoothness of streaming. However, these algorithms take similar actions as they take in the case of a large value of γ.

7 BOLA360 Enhancements

BOLA360 is meticulously designed to excel under all conceivable network conditions, including the most challenging worst-case-like scenarios. The aim to achieve a satisfactory performance across all input, however, makes BOLA360 often operate conservatively, refraining from switching to higher bitrates in many real-world situations where worst-case conditions fail to materialize. In this section, we propose BOLA360-REP and BOLA360-PL, two heuristic algorithms to improve the practical performance of BOLA360 could be improved from two perspectives. First, we introduce BOLA360-PL to address the common drawback of buffer-based ABR algorithms in fetching low-quality bitrates during start or seek time or high oscillations time intervals. Secondly, we propose `BOLA360-REP` to add the tile upgrade into the `BOLA360`. The basic `BOLA360` algorithm is not designed to replace previously downloaded tiles with higher bitrates, further restricting its adaptability.

`BOLA360-PL` is a generalized version of `BOLA-PL` introduced in [34]. It aims to reduce the reaction time of the `BOLA360` during start and seek times by virtually increasing the buffer level at the start or seek time. The reaction time is the duration from when the first tile is fetched (during start time) or the first seek tile is fetched (during seek time) until bitrate of selected tiles stabilizes. `BOLA360-PL` estimates the bandwidth and multiplies it by 50% to establish a safe expected bandwidth. To prevent rebuffering, `BOLA360-PL` limits the bitrate of each tile based on the estimated bandwidth throughput. More specifically, it restricts the size of the entire chunk to $S_{lim} = Q(t)w_p(t)/2D$, where $w_p(t)$ denotes the predicted bandwidth capacity at time t. `BOLA360-PL` virtually inserts a proportional number of tiles into the buffer such that the size of the new downloading chunk does not exceed S_{lim}.

One of the primary limitations of the basic version of the `BOLA360` algorithm is its inability to modify previously downloaded tiles. Specifically, `BOLA360` must make decisions about tiles of future chunks, and it cannot replace higher bitrate tiles with previously downloaded, lower quality ones. If the bandwidth capacity experiences a short-term decrease, `BOLA360` adjusts the download bitrates to match the new bandwidth capacity by switching to lower bitrates. When the bandwidth increases again, `BOLA360` may have already downloaded several tiles with low bitrates, and it cannot replace them with higher quality ones, even if the buffer level and bandwidth capacity are high. Consequently, `BOLA360` cannot utilize the entire bandwidth opportunity to optimize QoE. This challenge is addressed by the heuristic called `BOLA360-REP`.

`BOLA360-REP` is a variant of `BOLA360` that allows for the modification of previously downloaded tiles. Specifically, `BOLA360-REP` determines whether to download tiles for the next chunk or improve the quality of previously downloaded tiles, depending on the available video length in the buffer. `BOLA360-REP` uses a danger threshold set at 2δ and downloads tiles for a new chunk if the available video length in the buffer, $Q_{avl}(t)$, is below this threshold. If $Q_{avl}(t)$ exceeds the danger threshold, `BOLA360-REP` replaces previously downloaded tiles with higher bitrates.

When deciding to download tiles for a new chunk, `BOLA360-REP` selects bitrates according to `BOLA360`'s decisions. If the decision is to replace previously downloaded tiles, `BOLA360-REP` identifies tiles where there is at least a two-level difference between the current bitrate and what `BOLA360` would select at the current buffer level. `BOLA360-REP` then downloads and replaces those low-quality tiles. If no low-quality tiles are detected, `BOLA360-REP` proceeds to download tiles for the next chunk as usual. Overall, `BOLA360-REP` addresses the limitations of `BOLA360` and enhances the quality of experience for users.

7.1 Experimental Setup

We use the parameters from Section 6.4 and the head position probability profile 2 defined therein to evaluate the performance of the heuristic extensions, `BOLA360-PL` and `BOLA360-REP`. These algorithms are tested under two scenarios: 1) accurate head position probability predictions, and 2) noisy predictions for future chunks. In the first scenario, the head position probabilities provided to the ABR algorithms match the actual head position distribution of the user. This implies that the algorithm has perfect knowledge of the user's head position distribution, even for chunks that will be played far in the future.

In contrast, the second scenario assumes a 10% error in the prediction of head position probabilities for every δ seconds difference between the current chunk and the chunk for which the ABR algorithm is trying to predict head position probabilities. If the prediction error exceeds 100%, the head position prediction is considered unreliable, and the ABR algorithms are instead provided with uniform head position probabilities, where $p_{k,d} = 1/D$. In this case, if the ABR algorithm maintains a high buffer

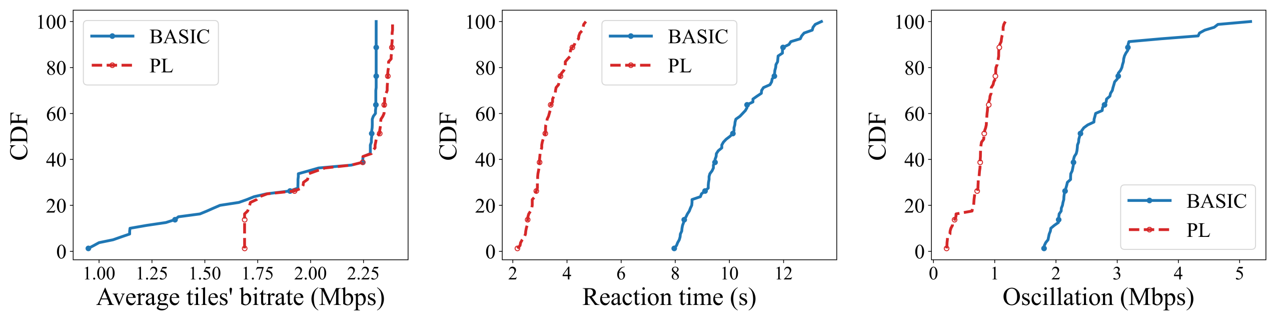


Figure 15: The CDF of the average bitrate of any downloaded tile (left), reaction time (middle), and oscillation (right) of basic `BOLA360` and `BOLA360-PL` using real network and head movement traces. `BOLA360-PL` reduces the oscillation and reaction time by 70.9% and 67.8% respectively.

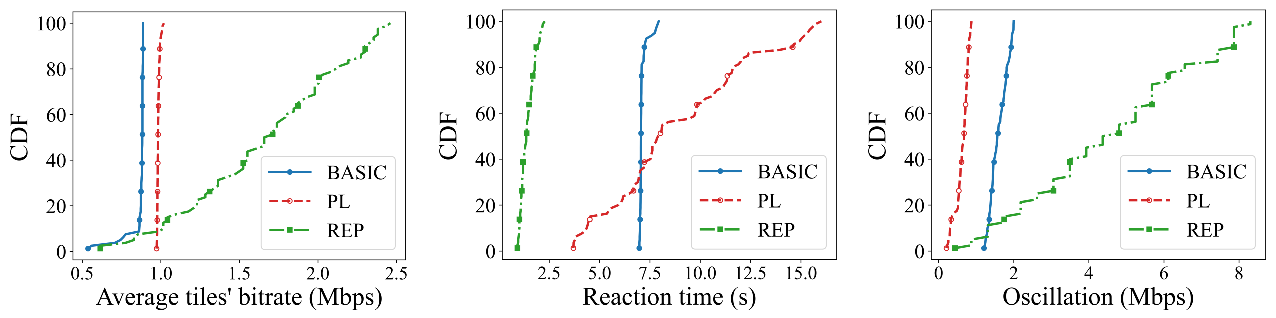


Figure 16: The CDF of average bitrate of downloaded tiles (left), reaction time (middle), and oscillation (right) of basic `BOLA360`, `BOLA360-PL`, and `BOLA360-REP` using real network and head movement traces while the prediction of the head position dynamically got updated. `BOLA360-REP` improves the average bitrate of downloaded tiles up to 91.2% compared to `BOLA360` BASIC, and reduces the reaction time by 80.0%.

utilization, there would be a long period of time between download time and playback time of each chunk. This long period leads to noisy prediction of head position probabilities at the time of download, resulting in reduced QoE for the user.

7.2 Experimental Results

Figure 15 shows the CDF plots of the average tiles' bitrate (left), the reaction time (middle), and oscillation (right, the average difference between the bitrate of two consecutive chunk for each tile) of 100 trials for accurate head position probability predictions. The results show that `BOLA360-PL` significantly reduces the oscillation and reaction time of `BOLA360`. Since the `BOLA360-PL` improves the bitrate of tiles during start and seek time, and these tiles are a low fraction of the entire video, the average bitrate of tiles that `BOLA360-PL` prepared for the user is slightly better than the average bitrate of tiles `BOLA360` downloads.

In Figure 16, we report the result of the evaluation of `BOLA360` and heuristic versions against the noisy prediction of head positions. Specifically, we report the CDF plot of the average tiles' bitrate, reaction time, and the oscillation of `BOLA360`, `BOLA360-PL`, and `BOLA360-REP`. The results show that the average bitrate of `BOLA360` and `BOLA360-PL` reduced compared to the case where accurate head position probabilities were available. On the other hand, `BOLA360-REP` improves the average bitrate of `BOLA360` up to near 97.6% and reduces the reaction time of `BOLA360` by 80.0%. Although `BOLA360-REP` could improve the average bitrate and the reaction time, it increases the oscillation. The average oscillation time for BOLA360 was 1.6 seconds, while this value for BOLA360-REP was 4.5 seconds. Meanwhile, all two heuristic versions could keep the rebuffering as low as the rebuffering of BOLA360.

Key takeaway. Each extension of BOLA360 improves the performance in certain aspects, such as bitrate or reaction time. However, each version has drawbacks that may result in lower performance in other aspects. Therefore, no version outperforms the others in all aspects, and depending on the application and user requirements, different versions may be suitable.

8 Related Work

The prior literature extensively addresses the problem of bitrate and view adaptation in 360° video streaming. Previous works commonly employ various machine learning techniques to predict user head movements and network throughput, incorporating these predictions into ABR algorithms [52, 31, 53, 54, 55, 56, 57, 58, 59, 60]. For example, [31] proposes a prediction-based approach and designs an ABR algorithm using historical data from 360° video streaming sessions. The focus of their work is on head movement prediction, while the ABR algorithm itself is a heuristic approach lacking rigorous optimization-based mechanisms.

Authors in [61] propose a Lyapunov-based model to solve the ABR360 problem, also utilized in [62]. They employ Lyapunov optimization for selecting and adjusting the bitrate of tiles, resulting in a nearly optimal ABR algorithm achieved through iterative updates to the tiles of a chunk. However, despite its near-optimality, this technique may suffer from a long reaction time and high wasted bandwidth due to iterative bitrate adjustment. In another work, [63] proposes a different approach by constructing a two-layered hierarchical buffer-based algorithm with short and long buffer layers. The prediction of FOV is used to perform short-term improvement. The long buffer layer tries to download the new tiles that are not available in the short buffer layer and will be played later. In another work, [64] predicts the head movement by using a saliency map, tile probability heat map, and LSTM models and gives ABR360 algorithm based on that.

In another category of work [65, 66, 67, 68, 17, 69], deep RL-based algorithms are developed for solving ABR360. They also use a dataset of the user's head position to train the model and find the optimal bitrate selection according to the predicted FOV.

In a recent study [70], the authors proposed a non-uniform coding and transmission method that divides the entire 360-degree field into four regions: an attention area, an out-of-field-of-view area, a peripheral area, and other areas. They then introduced an online ABR algorithm that dynamically selects appropriate bitrates for each region. This work was further extended by another study [52] that optimized the transmission strategy for 360-degree videos using LSTM networks.

In [71], FOV prediction is used to select proper bitrates for tiles in a predicted FOV, with the accuracy of prediction impacting the final bitrate selection. Other works such as [72, 30, 40, 73, 29] also focus on FOV prediction. The main idea is that users have similar region-of-interest when watching the same video. They divide the users into clusters such that users inside each cluster have similar region-of-interest in most videos. Then they give FOV prediction based on the cluster of a given user and the historical head direction traces of users in a predicted cluster. While these approaches help reduce bandwidth waste, they still require an ABR algorithm to select bitrates within the predicted region. In contrast, BOLA360 is an online algorithm with rigorous performance guarantees, solving the ABR360 problem optimally. Guan et al. [32] employ Model Predictive Control (MPC) to select the aggregate bitrate for a chunk, allocating it among tiles to maintain quality within the limited bitrate. In another category of research [74, 75], an optimized coding/encoding algorithm minimizes bandwidth usage for 360° videos, evaluated using real 4K and 8K videos from YouTube. Their experiments use a straightforward ABR algorithm resembling ProbDASH (Section 6).

9 Conclusion and Future Directions

In this paper, we formulated an optimization problem to maximize users' QoE in 360° video streaming applications. We proposed BOLA360, an online algorithm that achieves a provably near-optimal solution by selecting a proper bitrate for each tile of a 360° video to maximize quality while minimizing rebuffering rate. Our experimental results demonstrate that BOLA360 outperforms several alternative algorithms across various network and head movement profiles. In future work, we aim to develop a data-driven and robust version of BOLA360 that explicitly uses future predictions in decision-making while maintaining the algorithm's theoretical performance guarantees.

Acknowledgments

This research was supported in part by NSF grants CAREER 2045641, CPS-2136199, CNS-2106299, CNS-2102963, CSR-1763617, CNS-2106463, and CNS-1901137. We acknowledge their financial assistance in making this project possible.

References

[1] Youtube. youtube360. https://www.youtube.com/360, 2022. Accessed: 2022-03.

[2] Sony. Sony playstaion vr. https://www.playstation.com/en-us/ps-vr2/, 2022. Accessed: 2022-03.

[3] Google LLC. Google ar/vr. https://arvr.google.com/ar/, 2022. Accessed: 2022-03.

[4] CISCO. Cisco mobile visual networking index (vni) forecast projects 7-fold increase in global mobile data traffic from 2016-2021. https://tinyurl.com/CICSO-netwok, 2017. Accessed: 2022-08.

[5] Michael Zink, Ramesh Sitaraman, and Klara Nahrstedt. Scalable 360° video stream delivery: Challenges, solutions, and opportunities. *Proceedings of the IEEE*, 107(4):639–650, 2019.

[6] Thanh Cong Nguyen and Ji-Hoon Yun. Predictive tile selection for 360-degree vr video streaming in bandwidth-limited networks. *IEEE Communications Letters*, 22(9):1858–1861, 2018.

[7] Simone Mangiante, Guenter Klas, Amit Navon, Zhuang GuanHua, Ju Ran, and Marco Dias Silva. Vr is on the edge: How to deliver 360 videos in mobile networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 30–35, 2017.

[8] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM*, pages 1977–1986. IEEE, 2020.

[9] Akamai. Akamai's [state of the internet]. https://tinyurl.com/Akmai-internet-connectivity, 2017. Accessed: 2022-07.

[10] EtiSoftware. Internet speed and subscriber dissatisfaction. https://tinyurl.com/network-speed, 2021. Accessed: 2022-07.

[11] Jonathan Kua, Grenville Armitage, and Philip Branch. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys & Tutorials*, 19(3):1842–1866, 2017.

[12] Bo Han, Yu Liu, and Feng Qian. ViVo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th MobiCom*, pages 1–13, 2020.

[13] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of ACM SIGCOMM*, pages 197–210, 2017.

[14] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of ACM SIGCOMM*, pages 325–338, 2015.

[15] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In *NSDI*, pages 303–320, 2021.

[16] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.

[17] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of ACM SIGCOMM*, pages 107–125, 2020.

[18] Ehab Ghabashneh, Chandan Bothra, Ramesh Govindan, Antonio Ortega, and Sanjay Rao. Dragonfly: Higher perceptual quality for continuous 360 video playback. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 516–532, 2023.

[19] Chenge Li, Weixi Zhang, Yong Liu, and Yao Wang. Very long term field of view prediction for 360-degree video streaming. In *IEEE MIPR*, pages 297–302. IEEE, 2019.

[20] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th NOSSDAV*, pages 67–72, 2017.

[21] Yanyu Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, Jingyi Yu, and Shenghua Gao. Gaze prediction in dynamic 360 immersive videos. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5333–5342, 2018.

[22] Mai Xu, Yuhang Song, Jianyi Wang, MingLang Qiao, Liangyu Huo, and Zulin Wang. Predicting head movement in panoramic video: A deep reinforcement learning approach. *IEEE transactions on pattern analysis and machine intelligence*, 41(11):2693–2708, 2018.

[23] Yixuan Ban, Lan Xie, Zhimin Xu, Xinggong Zhang, Zongming Guo, and Yue Wang. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *IEEE ICME*, pages 1–6. IEEE, 2018.

[24] Zhimin Xu, Yixuan Ban, Kai Zhang, Lan Xie, Xinggong Zhang, Zongming Guo, Shengbin Meng, and Yue Wang. Tile-based QoE-driven HTTP/2 streaming system for 360 video. In *ICMEW*, pages 1–4. IEEE, 2018.

[25] Xianglong Feng, Yao Liu, and Sheng Wei. Livedeep: Online viewport prediction for live virtual reality streaming using lifelong deep learning. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 800–808. IEEE, 2020.

[26] Xianglong Feng, Weitian Li, and Sheng Wei. Liveroi: region of interest analysis for viewport prediction in live mobile virtual reality streaming. In *Proceedings of the 12th ACM MMSys*, pages 132–145, 2021.

[27] Ali Zeynali, Mohammad H Hajiesmaili, and Ramesh K Sitaraman. BOLA360: Near-optimal View and Bitrate Adaptation for 360-degree Video Streaming. In *Proceedings of the 15th ACM Multimedia Systems Conference*, pages 12–22, 2024.

[28] Cagri Ozcinar, Ana De Abreu, and Aljosa Smolic. Viewport-aware adaptive 360 video streaming using tiles for virtual reality. In *IEEE ICIP*, pages 2174–2178. IEEE, 2017.

[29] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 360probdash: Improving qoe of 360 video streaming using tile-based HTTP adaptive streaming. In *Proceedings of the 25th ACM MM*, pages 315–323, 2017.

[30] Shibo Wang, Shusen Yang, Hailiang Li, Xiaodan Zhang, Chen Zhou, Chenren Xu, Feng Qian, Nanbin Wang, and Zongben Xu. SalientVR: saliency-driven mobile 360-degree video streaming with gaze information. In *Proceedings of MobiCom*, pages 542–555, 2022.

[31] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of ACM MobiCom*, pages 99–114, 2018.

[32] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proceedings of the ACM SIGCOMM*, pages 394–407. 2019.

[33] Sohee Park, Arani Bhattacharya, Zhibo Yang, Samir R Das, and Dimitris Samaras. Mosaic: Advancing user quality of experience in 360-degree video streaming with machine learning. *IEEE Transactions on Network and Service Management*, 18(1):1000–1015, 2021.

[34] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2s):1–29, 2019.

[35] Joshua Ratcliff, Alexey Supikov, Santiago Alfaro, and Ronald Azuma. ThinVR: Heterogeneous microlens arrays for compact, 180 degree FOV VR near-eye displays. *IEEE transactions on visualization and computer graphics*, 26(5):1981–1990, 2020.

[36] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1161–1170. IEEE, 2016.

[37] Jinyu Chen, Xianzhuo Luo, Miao Hu, Di Wu, and Yipeng Zhou. Sparkle: User-aware viewport prediction in 360-degree video streaming. *IEEE Transactions on Multimedia*, 23:3853–3866, 2020.

[38] Jounsup Park and Klara Nahrstedt. Navigation graph for tiled media streaming. In *Proceedings of the 27th ACM MM*, pages 447–455, 2019.

[39] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6, 2016.

[40] Lan Xie, Xinggong Zhang, and Zongming Guo. Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *Proceedings of the 26th ACM MM*, pages 564–572, 2018.

[41] Michael J Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.

[42] Michael J Neely. Dynamic optimization and learning for renewal systems. *IEEE Transactions on Automatic Control*, 58(1):32–46, 2012.

[43] Peter Reichl, Bruno Tuffin, and Raimund Schatz. Logarithmic laws in service quality perception: where microeconomics meets psychophysics and quality of experience. *Telecommunication Systems*, 52(2):587–600, 2013.

[44] Han Hu, Cheng Zhan, Jianping An, and Yonggang Wen. Optimization for HTTP adaptive video streaming in UAV-enabled relaying system. In *ACM ICC*, pages 1–6. IEEE, 2019.

[45] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[46] Mingyue Hao, Jinghao Yuan, Bingcong Lu, Li Song, Rong Xie, and Wenjun Zhang. Buffer Displacement Based Online Learning Algorithm For Low Latency HTTP Adaptive Streaming. In *2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE, 2021.

[47] A Bokani, M Hassan, SS Kanhere, J Yao, and G Zhong. Comprehensive mobile bandwidth traces from vehicular networks. In *Proceedings of the 7th ACM MMSys*, pages 1–6, 2016.

[48] Ghent University. 4G/LTE Bandwidth Logs. `https://users.ugent.be/~jvdrhoof/dataset-4g/`, 2019. Accessed: 2022-06.

[49] 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*, 20(11):2177–2180, 2016.

[50] Google LLC. Google-chrome: Chrome devtools protocol. `https://chromedevtools.github.io/devtools-protocol/tot/Network/`, 2023. Accessed: 2023-01.

[51] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A dataset for exploring user behaviors in VR spherical video streaming. In *MMSys*, pages 193–198, 2017.

[52] Jia Guo, Chengrui Li, Jinqi Zhu, Xiang Li, Qian Gao, Yunhe Chen, and Weijia Feng. Long Short-Term Memory-Based Non-Uniform Coding Transmission Strategy for a 360-Degree Video. *Electronics*, 13(16):3281, 2024.

[53] Jie Li, Ling Han, Chong Zhang, Qiyue Li, and Zhi Liu. Spherical convolution empowered viewport prediction in 360 video multicast with limited FoV feedback. *ACM Transactions on Multimedia Computing, Communications and Applications*, 19(1):1–23, 2023.

[54] Yucheng Zhu, Xiongkuo Min, Dandan Zhu, Guangtao Zhai, Xiaokang Yang, Wenjun Zhang, Ke Gu, and Jiantao Zhou. Toward visual behavior and attention understanding for augmented 360 degree videos. *ACM Transactions on Multimedia Computing, Communications and Applications*, 19(2s):1–24, 2023.

[55] Yumei Wang, Junjie Li, Zhijun Li, Simou Shang, and Yu Liu. Synergistic Temporal-Spatial User-Aware Viewport Prediction for Optimal Adaptive 360-Degree Video Streaming. *IEEE Transactions on Broadcasting*, 2024.

[56] Zhiahao Zhang, Yiwei Chen, Weizhan Zhang, Caixia Yan, Qinghua Zheng, Qi Wang, and Wangdu Chen. Tile classification based viewport prediction with multi-modal fusion transformer. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 3560–3568, 2023.

[57] Mehdi Setayesh and Vincent WS Wong. A Content-based Viewport Prediction Framework for 360° Video Using Personalized Federated Learning and Fusion Techniques. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*, pages 654–659. IEEE, 2023.

[58] Qin Yang, Wenxuan Gao, Chenglin Li, Hao Wang, Wenrui Dai, Junni Zou, Hongkai Xiong, and Pascal Frossard. 360Spred: Saliency Prediction for 360-Degree Videos Based on 3D Separable Graph Convolutional Networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2024.

[59] Zhaolin Wan, Han Qin, Ruiqin Xiong, Zhiyang Li, Xiaopeng Fan, and Debin Zhao. Predicting 360° Video Saliency: A ConvLSTM Encoder-Decoder Network with Spatio-temporal Consistency. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[60] Abid Yaqoob and Gabriel-Miro Muntean. A Collaborative Trajectory-Oriented Viewport Prediction for on-Demand and Live 360 VR Video Streaming. In *2023 IEEE 20th International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pages 1–6. IEEE, 2023.

[61] Haodan Zhang, Yixuan Ban, Zongming Guo, Ken Chen, and Xinggong Zhang. RAM360: Robust Adaptive Multi-layer 360 Video Streaming with Lyapunov Optimization. *IEEE Transactions on Multimedia*, 2023.

[62] Wang Shen, Lianghui Ding, Guangtao Zhai, Ying Cui, and Zhiyong Gao. A QoE-oriented saliency-aware approach for 360-degree video transmission. In *IEEE VCIP*, pages 1–4. IEEE, 2019.

[63] Zhiqian Jiang, Xu Zhang, Wei Huang, Hao Chen, Yiling Xu, Jenq-Neng Hwang, Zhan Ma, and Jun Sun. A hierarchical buffer management approach to rate adaptation for 360-degree video streaming. *IEEE Transactions on Vehicular Technology*, 69(2):2157–2170, 2019.

[64] Sohee Park, Arani Bhattacharya, Zhibo Yang, Mallesham Dasari, Samir R Das, and Dimitris Samaras. Advancing user quality of experience in 360-degree video streaming. In *IFIP Networking*. IEEE, 2019.

[65] Yuanxing Zhang, Pengyu Zhao, Kaigui Bian, Yunxin Liu, Lingyang Song, and Xiaoming Li. DRL360: 360-degree video streaming with deep reinforcement learning. In *IEEE INFOCOM*, pages 1252–1260. IEEE, 2019.

[66] Sohee Park, Minh Hoai, Arani Bhattacharya, and Samir R Das. Adaptive streaming of 360-degree videos with reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1839–1848, 2021.

[67] Jun Fu, Chen Hou, and Zhibo Chen. 360hrl: Hierarchical Reinforcement Learning Based Rate Adaptation for 360-Degree Video Streaming. In *IEEE VCIP*, pages 1–5. IEEE, 2021.

[68] Nuowen Kan, Junni Zou, Chenglin Li, Wenrui Dai, and Hongkai Xiong. RAPT360: Reinforcement learning-based rate adaptation for 360-degree video streaming with adaptive prediction and tiling. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(3):1607–1623, 2021.

[69] Haopeng Wang, Zijian Long, Haiwei Dong, and Abdulmotaleb El Saddik. MADRL-Based Rate Adaptation for 360 Video Streaming With Multi-Viewpoint Prediction. *IEEE Internet of Things Journal*, 2024.

[70] Jia Guo, Shiqiang Li, Jinqi Zhu, Xiang Li, Bowen Sun, and Weijia Feng. Adaptive Transmission Strategy for Non-Uniform Coding of 360 Videos. *Electronics (2079-9292)*, 13(16), 2024.

[71] Hui Yuan, Shiyun Zhao, Junhui Hou, Xuekai Wei, and Sam Kwong. Spatial and temporal consistency-aware dynamic adaptive streaming for 360-degree videos. *IEEE Journal of Selected Topics in Signal Processing*, 14(1):177–193, 2019.

[72] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM MM*, pages 1190–1198, 2018.

[73] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An HTTP/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of ACM MM*, pages 306–314, 2017.

[74] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th ACM MobiSys*, pages 482–494, 2018.

[75] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *NSDI*, pages 953–971, 2023.