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

Ali Zeynali
azeynali@cs.umass.edu
University of Massachusetts Amherst

Mohammad H. Hajiesmaili
hajiesmaili@cs.umass.edu
University of Massachusetts Amherst

Ramesh K. Sitaraman
ramesh@cs.umass.edu
University of Massachusetts Amherst
& Akamai Technologies

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.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Theory of computation** → *Stochastic control and optimization.*

KEYWORDS

Online video streaming, quality of experience, bitrate selection, view adaptation, 360-degree video streaming

ACM Reference Format:
Ali Zeynali, Mohammad H. Hajiesmaili, and Ramesh K. Sitaraman. 2024. BOLA360: Near-optimal View and Bitrate Adaptation for 360-degree Video Streaming. In *ACM Multimedia Systems Conference 2024 (MMSys '24), April 15–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3625468.3647607

MMSys '24, April 15–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-0412-3/24/04
https://doi.org/10.1145/3625468.3647607

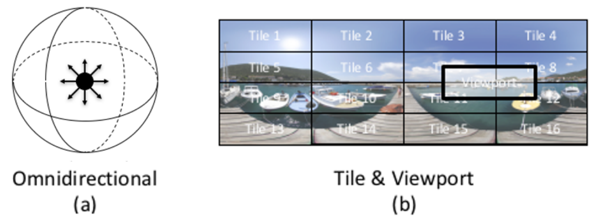

Figure 1: (a) Users watch 360° videos by moving their viewport to point to any direction in the enclosing sphere (b) each frame of the 360° video is broken up into tile frames [63].

1 INTRODUCTION

With recent advancements in omnidirectional cameras and AR/VR headsets, users can enjoy 360° media like YouTube 360 [58], virtual and augmented reality applications [45, 25].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 [6]. 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 [63], 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

[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`.

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 [32, 27, 7]. 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 [27]. 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 [1, 8]. 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 degraded quality-of-experience, or even cybersickness [63].

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 [23, 15, 28, 57, 61, 46, 22], 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 [13].

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[24, 9, 55, 54, 2, 56, 10, 11]. 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 BOLA360, 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.

3) We analyze BOLA360's performance, demonstrating that (i) it never exceeds the client's buffer capacity (Theorem 4.1) and (ii) its average QoE is within a small additive constant factor of the offline optimum of ABR360 (Theorem 4.2), 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 1).

4) 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 [33], ProbDASH [52], Salient-VR [50], Flare [39], Pano [14], and Mosaic [37]. 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 6) and 9% improvements over 12 different head position probability distributions (Figure 8).

5) Finally, we explore two extensions to BOLA360, addressing specific real-world scenarios [47]. 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 BOLA360-PL and BOLA360-REP, each targeting specific limitations of the original algorithm. Our experiments showBOLA360-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. In Section 4, we develop BOLA360 using a Lyapunov optimization approach, proving its near-optimality. Section 5 evaluates the performance BOLA360 against state-of-the-art algorithms. In Section 6, we introduce two enhancements for BOLA360 which practically improves its performance. The related work is discussed in Section 7, and we conclude in Section 8.

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°[41]. Therefore, the user interacting with the 360° video cannot see the entire spatial area of the presented video. The part of the

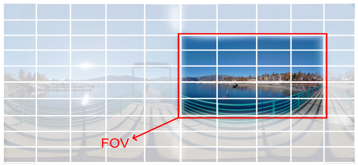

Figure 2: One shot from the entire spatial area of 360° video and FOV of user in that

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 [47] 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

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 [3, 24, 5, 50, 34], or from a chunk analysis of the content combined with points probability analysis of 360° sphere [52, 40, 53]. 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.

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

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.

4 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.

4.1 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 1.

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.

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

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

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

where $a(k)$ 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 $\mathbf{a}(k) = [a_{k,1,1}, a_{k,1,2}, ..., a_{k,1,M}, a_{k,2,1}, ..., a_{k,D,M}]$ 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 [46], 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 1: BOLA360 (k)

```
a(k): solution of optimization problem (5);
if number of non-zero elements in a(k) > 0 then
    download bitrates for chunk k according to a(k);
end
else
    wait for Δ seconds and repeat the bitrate selection for
      this chunk again;
end
```

4.2 Theoretical Analysis of BOLA360

We first provide an upper bound for the buffer level of BOLA360 in Theorem 4.1. Second, in Theorem 4.2, 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 1.

THEOREM 4.1. *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 [46]. 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 $V\delta(v_M + \gamma\delta) + D\delta$.

Now, we proceed to analyze the QoE of BOLA360. With large K, the ABR360 problem with rate stability constraint [30] is equivalent to the relaxed version of ABR360 with limited buffer capacity, i.e.,

$$Q(t_k) \leq Q_{\max}$$
$$\Rightarrow \lim_{K\to\infty} \frac{1}{K}\mathbb{E}\left\{\sum_{k=1}^{K}\sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\delta\right\} \leq \lim_{K\to\infty} \frac{1}{K}\mathbb{E}\left\{\sum_{k=1}^{K} n_k T_k\right\}.$$

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 [29]. 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 [29], the offline ABR360 problem can convert into an optimization problem over renewal frames. Then, following Lemma 1 in [29], 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.,* $K \longrightarrow \infty$*, 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 [29] and continues with the approach taken for proof of Lemma 1 in [46]. 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 [29].

Theorem 4.2 (main theorem). *Let* OBJ *be the expected QoE achieved by* BOLA360*. For a large video, i.e.,* $K \longrightarrow \infty$,

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

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

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

Regardless of the buffer level, there would be an upper bound for the value of $\Phi(t_k)$ using the buffer level evolution described in Equation (4).

$$\Phi(t_k) \le \frac{D\delta^2 + \Psi}{2\delta^2} - \frac{Q(t_k)}{\delta}\mathbb{E}\left\{\frac{n_k T_k}{\delta} - \sum_{d=1}^{D}\sum_{m=1}^{M} a_{k,d,m}\}|Q(t_k)\right\},$$

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

The previous equation holds since the decision of BOLA360 at time t_k is a solution of the maximization equation detailed in Equation (5). Then, we have

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

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,

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

By dividing all terms by $V \times K \times \mathbb{E}\{T_k\}$ and taking the limit $K \longrightarrow +\infty$, the proof completes.

Remark 1 (On the conflict between the playback delay and QoE of streaming). *Theorem 4.2 states as the value of* V *increases, the performance of* BOLA360 *gets closer to the optimal QoE. However, Theorem 4.1 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.*

4.3 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 [42, 46, 18]. 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. 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 useris 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

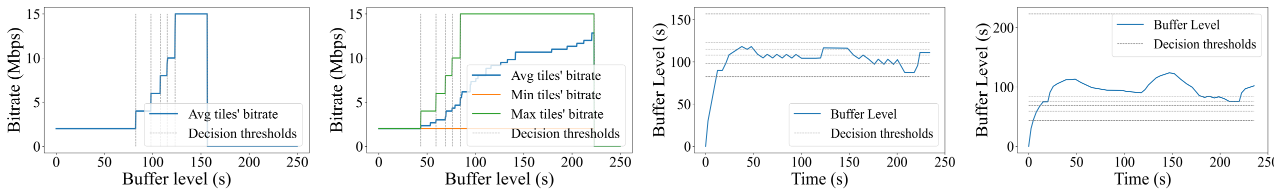

Figure 3: The selected bitrate of BOLA360 for tiles with highest and lowest probability and average selected bitrate as a function of buffer level for homogeneous (leftmost) and heterogeneous (second left) distributions, and buffer level variation over time for homogeneous (third left) and heterogeneous (rightmost) distributions.

($p_{k,d} = 1/D$ 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, respectively. 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 3 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 3. When the buffer level is higher 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 4 shows 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.

5 COMPARISON ALGORITHMS

We compare BOLA360 with VA-360 [33], ProbDASH [52], Flare [39], Salient-VR [50], Pano [14], and Mosaic [37], 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 [57] 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.

5.1 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 4.3, we use a logarithmic utility function. 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 [19, 16] to ensure fairness. We employ dynamic value selection for Δ, as proposed in [46], 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 [46]. 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 [4] 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 [26] 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 [34] for 360° video using public VR head traces [51].

5.2 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 [4] for this comparison. We report playing bitrate, the rebuffering ratio (percentage of length of video considered as a rebuffering), and QoE of BOLA360, and state-of-the-art algorithms. Note that the average playing bitrate reported in Figure 5 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 QoE values of 100 different trials is reported in Figure 5.

The results in Figure 5 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

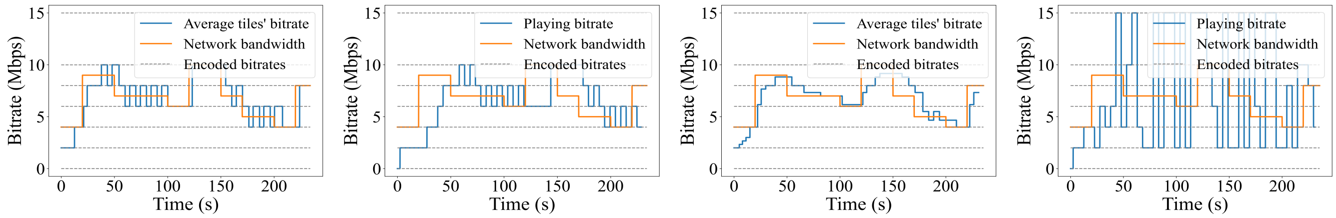

Figure 4: Variation of average downloaded bitrates and playing bitrate over time under bitrate selection of BOLA360 for the homogeneous (left most and second left), and heterogeneous (third left and right most) head position probability distribution.

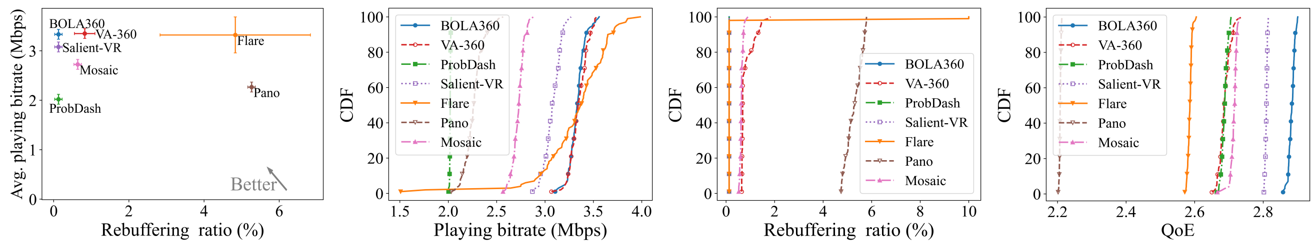

Figure 5: Average playing bitrate vs. rebuffering ratio (leftmost), the CDF of playing bitrate (second left), rebuffering ratio (third left), and QoE (rightmost) of BOLA360 and comparison algorithms using real network and head movement traces. The average playing bitrate of BOLA360 is $3.34 Mbps$**, 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%**.**

distributes the available bitrates among different tiles such that BOLA360 can achieve a lower rebuffering ratio.

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.

5.3 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 5.1) for all algorithms to capture the impact of the network capacity on their performance.

Figure 6 shows the average 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 bitrates compared to other algorithms, while its high rebuffering, shown in Figure 7, lowers the QoE of this algorithm. In addition, the playback delay of BOLA360 and comparison algorithms are shown in Figure 7. 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 4.1 and the fact that BOLA360 tries to keep the buffer level high.

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.

5.4 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 5.2. Specifically, for each chunk k, we replace the set of probabilities with the probabilities calculated from Equation (11).

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 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.

$$p_i(k) = \frac{1 - \alpha_p(k)}{D_{\text{pos}}(k)} + \alpha_p(k) p_i^{(L)}\Big(k, r(k)\Big), \qquad (11)$$

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 (11), $\alpha_p(k)$ 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

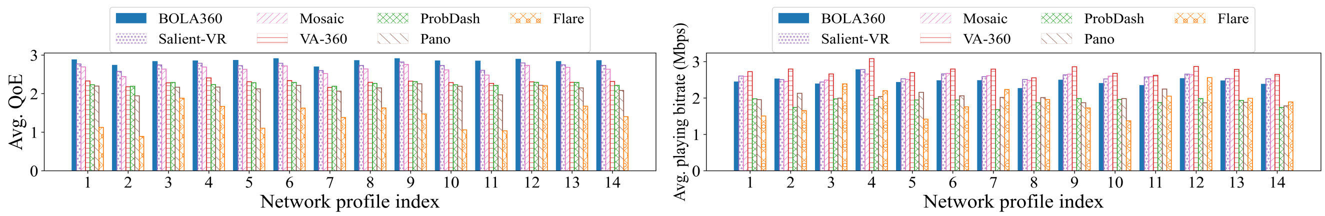

Figure 6: The average QoE (left) and average playing bitrate (right) over the bitrate selection of `BOLA360` and other comparison algorithms for 14 different network profiles and 100 trials. In terms of QoE, `BOLA360` outperforms others in all profiles. On average, `BOLA360` provides about 6% **improvement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

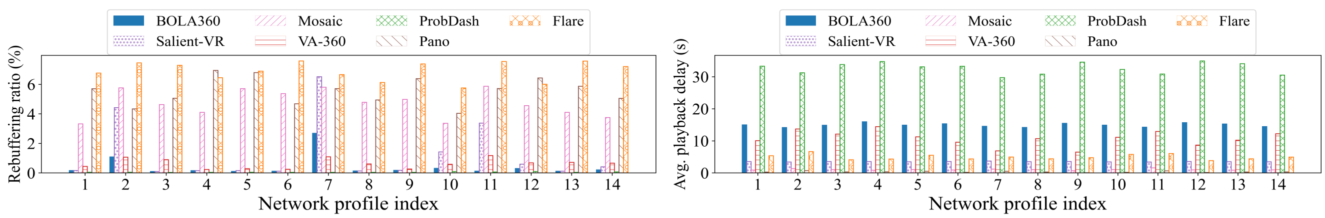

Figure 7: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of `BOLA360` and comparison algorithms for 14 different network profiles and 100 trials. `Pano` and `Flare` usually show higher rebuffering than the other algorithms, while their playback delay is shorter. The average playback delay for `BOLA360` is 14.9 **seconds.**

$r(k) = 0.05$ 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 1.

We report the average 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 8 (average QoE and playing bitrate), and 9 (average rebuffering ratio and playback delay). Figure 8 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.

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 `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 [47]. 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}.

The second heuristic, `BOLA360-REP`, allows for upgrading of previously downloaded tiles. `BOLA360-REP` determines whether it is better to download a new tile for the next chunk or to improve the quality of previously downloaded tiles based on the length of video available in the buffer. `BOLA360-REP` identifies a tile in the buffer where there is at least a two-level difference between the bitrate of downloaded tiles for that tile and the bitrate that `BOLA360` would select for that tile at the current time. If such a tile is identified, then the decision of `BOLA360-REP` is to upgrade the bitrate of those low-quality tiles. Otherwise, it downloads tiles for the next chunk according to the decision of `BOLA360`. To prevent upgrading tiles about to be played shortly, `BOLA360-REP` selectively upgrades tiles set to render at least 2δ seconds later.

Table 1: The details of the probability distributions used in the experiment of Section 5.4

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

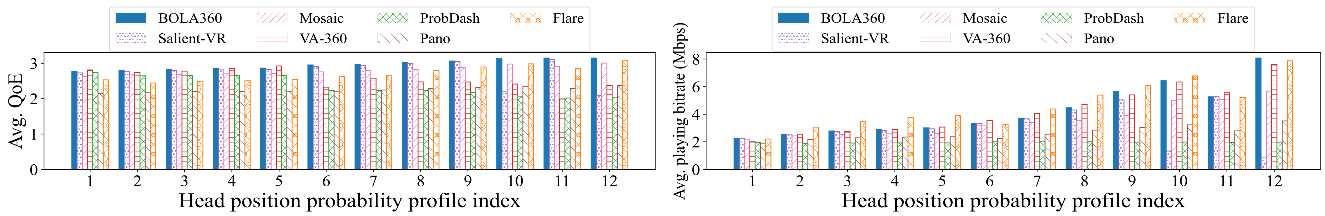

Figure 8: The average QoE (left) and average playing bitrate (right) over the bitrate selection of `BOLA360` and comparison algorithms using 12 different head position probability distributions over 100 trials. On average, `BOLA360` provides 9% improvement to the QoE of `Salient-VR`, and 31% to QoE of `Pano`.

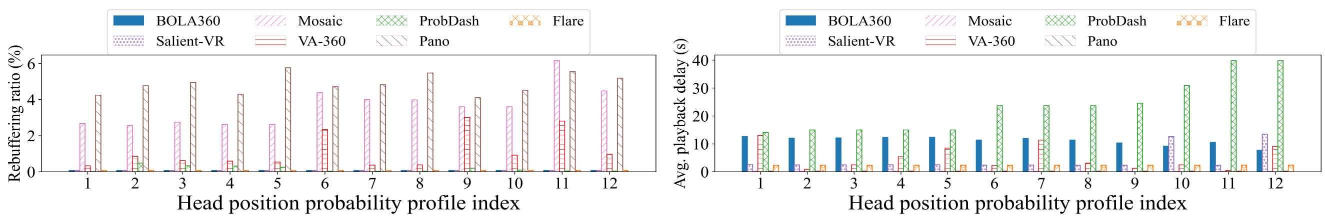

Figure 9: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of `BOLA360` and comparison algorithms using 12 different head position probability distributions over 100 trials. `VA-360` usually results in a high rebuffering ratio and short playback delay. 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.

6.1 Experimental Setup

We choose the parameters used in Section 5.4 and the head position probability profile 2 defined in that section to evaluate the performance of heuristic extensions `BOLA360-PL` and `BOLA360-REP`. We evaluate the performance of these algorithms against two scenarios: 1) accurate head position probability prediction; and 2) noisy prediction for future chunks. In the first scenario, the head position probabilities provided to the ABR algorithms are identical to the user's actual head position distribution. This means that the algorithm knows the user's head position distribution, even for tiles of chunks that will be played far in the future. In contrast, the second scenario assumes that there will be a 10% error added to the prediction of head position probabilities for every δ seconds difference between the chunk the user is watching and the chunk the ABR is seeking to obtain head position probabilities for. Note that if the error is greater than 100%, the prediction of head position is considered unavailable, and the head position probabilities passed to the ABR algorithms are uniform distributions, where $p_{k,d} = 1/D$.

6.2 Experimental Results

Figure 10 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 11, 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.

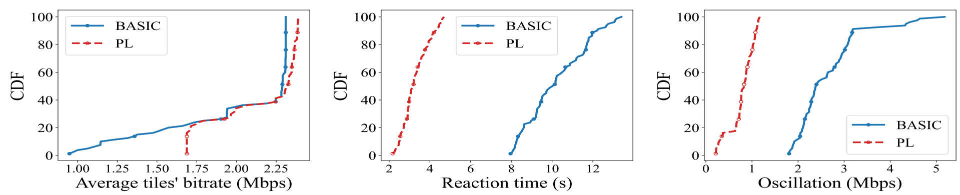

Figure 10: The CDF of the average bitrate of any downloaded tile (left), reaction time (middle), and oscillation (right) of basic `BOLA360` and `BOLA360-PL` using real network and head movement traces. `BOLA360-PL` reduces the oscillation and reaction time by 70.9% **and** 67.8% **respectively.**

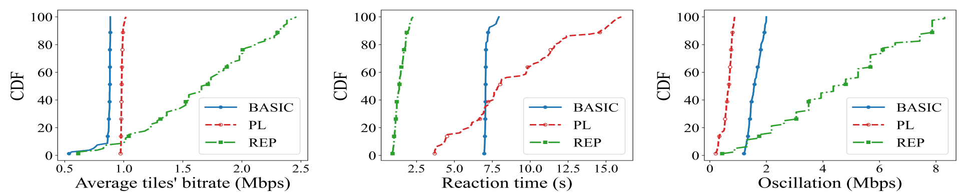

Figure 11: The CDF of average bitrate of downloaded tiles (left), reaction time (middle), and oscillation (right) of basic `BOLA360`, `BOLA360-PL`, and `BOLA360-REP` using real network and head movement traces while the prediction of the head position dynamically got updated. `BOLA360-REP` improves the average bitrate of downloaded tiles up to 91.2% **compared to `BOLA360` BASIC, and reduces the reaction time by** 80.0%**.**

7 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 incorporate them into existing ABR algorithms. For example, [39] 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 [60] propose a Lyapunov-based model to solve the `ABR360` problem, also utilized in [44]. 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, [20] 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, [36] 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 [62, 35, 12, 21, 22], 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 [59], `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 [31, 50, 53, 38, 52] 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. [14] 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 [17, 43], 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 5).

8 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 work is supported by the U.S. National Science Foundation (NSF) under grant numbers CAREER-2045641, CPS-2136199, CNS-2106299, CNS-2102963, CNS-1763617, CNS-1901137, and CNS-2106463.

REFERENCES

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

[2] Yixuan Ban et al. 2018. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *IEEE ICME*. IEEE, 1–6.

[3] Yanan Bao et al. 2016. Shooting a moving target: motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1161–1170.

[4] A Bokani et al. 2016. Comprehensive mobile bandwidth traces from vehicular networks. In *Proceedings of the 7th ACM MMSys*, 1–6.

[5] Jinyu Chen et al. 2020. Sparkle: user-aware viewport prediction in 360-degree video streaming. *IEEE Transactions on Multimedia*, 23, 3853–3866.

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

[7] Mallesham Dasari et al. 2020. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM*. IEEE, 1977–1986.

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

[9] Ching-Ling Fan et al. 2017. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th NOSSDAV*, 67–72.

[10] Xianglong Feng et al. 2020. Livedeep: online viewport prediction for live virtual reality streaming using lifelong deep learning. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 800–808.

[11] Xianglong Feng et al. 2021. Liveroi: region of interest analysis for viewport prediction in live mobile virtual reality streaming. In *Proceedings of the 12th ACM MMSys*, 132–145.

[12] Jun Fu et al. 2021. 360hrl: Hierarchical Reinforcement Learning Based Rate Adaptation for 360-Degree Video Streaming. In *IEEE VCIP*. IEEE, 1–5.

[13] Ehab Ghabashneh et al. 2023. Dragonfly: higher perceptual quality for continuous 360 video playback. In *Proceedings of the ACM SIGCOMM 2023 Conference*, 516–532.

[14] Yu Guan et al. 2019. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proceedings of the ACM SIGCOMM*, 394–407.

[15] Bo Han et al. 2020. ViVo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th MobiCom*, 1–13.

[16] Mingyue Hao et al. 2021. 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.

[17] Jian He et al. 2018. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th ACM MobiSys*, 482–494.

[18] Han Hu et al. 2019. Optimization for HTTP adaptive video streaming in UAV-enabled relaying system. In *ACM ICC*. IEEE, 1–6.

[19] Te-Yuan Huang et al. 2014. A buffer-based approach to rate adaptation: evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, 187–198.

[20] Zhiqian Jiang et al. 2019. A hierarchical buffer management approach to rate adaptation for 360-degree video streaming. *IEEE Transactions on Vehicular Technology*, 69, 2, 2157–2170.

[21] Nuowen Kan et al. 2021. 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.

[22] Jaehong Kim et al. 2020. Neural-enhanced live streaming: improving live video ingest via online learning. In *Proceedings of ACM SIGCOMM*, 107–125.

[23] Jonathan Kua et al. 2017. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys & Tutorials*, 19, 3, 1842–1866. DOI: 10.1109/COMST.2017.2685630.

[24] Chenge Li et al. 2019. Very long term field of view prediction for 360-degree video streaming. In *IEEE MIPR*. IEEE, 297–302.

[25] Google LLC. 2022. Google ar/vr. https://arvr.google.com/ar/. Accessed: 2022-03. (2022).

[26] Google LLC. 2023. Google-chrome: chrome devtools protocol. https://chromedevtools.github.io/devtools-protocol/tot/Network/. Accessed: 2023-01. (2023).

[27] Simone Mangiante et al. 2017. 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*, 30–35.

[28] Hongzi Mao et al. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of ACM SIGCOMM*, 197–210.

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

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

[31] Anh Nguyen et al. 2018. Your attention is unique: detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM MM*, 1190–1198.

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

[33] Cagri Ozcinar et al. 2017. Viewport-aware adaptive 360 video streaming using tiles for virtual reality. In *IEEE ICIP*. IEEE, 2174–2178.

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

[35] Sohee Park et al. 2021. Adaptive streaming of 360-degree videos with reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 1839–1848.

[36] Sohee Park et al. 2019. Advancing user quality of experience in 360-degree video streaming. In *IFIP Networking*. IEEE.

[37] Sohee Park et al. 2021. 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.

[38] Stefano Petrangeli et al. 2017. An HTTP/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of ACM MM*, 306–314.

[39] Feng Qian et al. 2018. Flare: practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of ACM MobiCom*, 99–114.

[40] Feng Qian et al. 2016. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 1–6.

[41] Joshua Ratcliff et al. 2020. ThinVR: Heterogeneous microlens arrays for compact, 180 degree FOV VR near-eye displays. *IEEE transactions on visualization and computer graphics*, 26, 5, 1981–1990.

[42] Peter Reichl et al. 2013. Logarithmic laws in service quality perception: where microeconomics meets psychophysics and quality of experience. *Telecommunication Systems*, 52, 2, 587–600.

[43] Michael Rudow et al. 2023. Tambur: efficient loss recovery for videoconferencing via streaming codes. In *NSDI*, 953–971.

[44] Wang Shen et al. 2019. A QoE-oriented saliency-aware approach for 360-degree video transmission. In *IEEE VCIP*. IEEE, 1–4.

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

[46] Kevin Spiteri et al. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28, 4, 1698–1711.

[47] Kevin Spiteri et al. 2019. From theory to practice: improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15, 2s, 1–29.

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

[49] J. van der Hooft et al. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20, 11, 2177–2180.

[50] Shibo Wang et al. 2022. SalientVR: saliency-driven mobile 360-degree video streaming with gaze information. In *Proceedings of MobiCom*, 542–555.

[51] Chenglei Wu et al. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *MMSys*, 193–198.

[52] Lan Xie et al. 2017. 360probdash: Improving qoe of 360 video streaming using tile-based HTTP adaptive streaming. In *Proceedings of the 25th ACM MM*, 315–323.

[53] Lan Xie et al. 2018. Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *Proceedings of the 26th ACM MM*, 564–572.

[54] Mai Xu et al. 2018. Predicting head movement in panoramic video: a deep reinforcement learning approach. *IEEE transactions on pattern analysis and machine intelligence*, 41, 11, 2693–2708.

[55] Yanyu Xu et al. 2018. Gaze prediction in dynamic 360 immersive videos. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5333–5342.

[56] Zhimin Xu et al. 2018. Tile-based QoE-driven HTTP/2 streaming system for 360 video. In *ICMEW*. IEEE, 1–4.

[57] Xiaoqi Yin et al. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of ACM SIGCOMM*, 325–338.

[58] Youtube. 2022. Youtube360. https://www.youtube.com/360. Accessed: 2022-03. (2022).

[59] Hui Yuan et al. 2019. Spatial and temporal consistency-aware dynamic adaptive streaming for 360-degree videos. *IEEE Journal of Selected Topics in Signal Processing*, 14, 1, 177–193.

[60] Haodan Zhang et al. 2023. RAM360: Robust Adaptive Multi-layer 360 Video Streaming with Lyapunov Optimization. *IEEE Transactions on Multimedia*.

[61] Xu Zhang et al. 2021. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In *NSDI*, 303–320.

[62] Yuanxing Zhang et al. 2019. DRL360: 360-degree video streaming with deep reinforcement learning. In *IEEE INFOCOM*. IEEE, 1252–1260.

[63] Michael Zink et al. 2019. Scalable 360° video stream delivery: challenges, solutions, and opportunities. *Proceedings of the IEEE*, 107, 4, 639–650.