

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

Ali Zeynali* Mohammad Hajiesmaili† Ramesh K. Sitaraman‡

September 11, 2023

Abstract

Recent advances in omnidirectional cameras and AR/VR headsets have spurred the adoption of 360° videos that 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 segments from the server so as to maximize the quality-of-experience (QoE) of the user. **BOLA360** conserves bandwidth by downloading only those video segments 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 segments 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 13.6% to 372.5% more QoE than 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.

arXiv:2309.04023v1 [cs.MM] 7 Sep 2023

*University of Massachusetts Amherst. Email: azeynali@cs.umass.edu.

†University of Massachusetts Amherst. Email: hajiesmaili@cs.umass.edu.

‡University of Massachusetts Amherst & Akamai Technologies. Email: ramesh@cs.umass.edu.

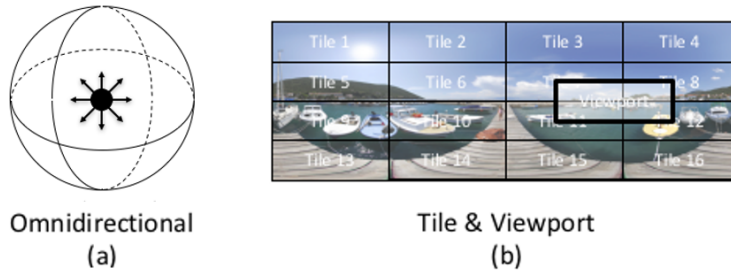


Figure 1: (a) Users watch 360° videos by moving their viewport to point to any direction in the enclosing sphere (b) the sphere is broken up into tiles and each tile in the user’s FOV is streamed as a sequence of segments [6].

1 Introduction

With recent advancements in omnidirectional cameras and AR/VR headsets, users can enjoy 360° media like YouTube 360 [1], virtual reality video games [2, 3], and augmented reality applications like Google AR/VR [4]. 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 driven in part 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 [5]. 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 [6], 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. The 360° videos are created and stored in servers and delivered to the users (i.e., client) in an *online* fashion. A common approach to 360° video delivery is to divide the viewing sphere of the user into a set of tiles (see Figure 1). Each tile is stored as a sequence of video segments, where each video segment can be played for a fixed duration of δ (say, $\delta = 5$) seconds. Each segment is encoded in multiple bitrates (i.e., resolutions) so that the quality of the segments 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 segments are streamed from the server ahead of time and buffered at the client before they can be rendered to the user. As the user changes their viewport, say by moving their head, the appropriate segments for the 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 a traditional (2D) videos [7, 8, 9]. The reason is that there are multiple tiles required to cover the 360° viewing sphere, with each tile encoded in multiple bitrates in a manner similar to a 2D video. Further, a high resolution of 4K to 8K is recommended for viewing AR/VR media [8]. Thus, the data rate of a 360° video that delivers a 4K stream to each eye (x2) and allows the user to watch the full 360° viewing sphere (x8 tiles, say) is 400 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 [10, 11]. Additionally, when the user’s viewport changes, say due to a head movement, the new segments 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 segment or no segment at all (i.e., blank screen). If

¹Field 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.

the “motion-to-photon” latency exceeds a few tens of milliseconds, the user experiences a degraded quality-of-experience, or even cybersickness [6].

Adaptive bitrate (ABR) algorithms for 360° videos. To ameliorate the challenge imposed by the size of the 360° videos, we study *adaptive bitrate (ABR) algorithms* that run on the client’s device and orchestrate the download of the video segments from the server. ABR algorithms for traditional (2D) videos is a well-studied problem with more than a decade of research [12, 13, 14, 15, 16, 17, 18]. In the traditional 2D video setting, ABR algorithms primarily adapt the bitrates of the downloaded segments to the available client-server bandwidth, ensuring that the video plays continuously without rebuffers (i.e., freezes). ABR algorithms for 360° videos are considerably more complex since they must *simultaneously* perform two types of adaptations. First, the algorithm must perform *view adaptation* by predicting ahead of time where the user’s head position might be and what tile(s) the user may view in the future. Second, the algorithm must perform *bitrate adaptation* by deciding what bitrates to download the segments of the predicted tiles. Importantly, these two types of adaptations must be optimized jointly since tiles that are more likely to be in the user’s viewport should be downloaded at higher bitrates.

Why naive ABR solutions do not work. A naive ABR algorithm divides the available client-server bandwidth equally among all tiles of the 360° video, resulting in downloading a segment for all tiles. While this prevents rebuffering since there is a downloaded segment for each tile, it leads to lower video quality and wasted segments that are never viewed. An alternative approach predicts the tile(s) the user is likely to watch and downloads segments only for those tile(s). This reduces the number of segments downloaded, allowing for higher quality. However, it is prone to rebuffering if the user watches unpredicted tile(s) for which no segments were downloaded [19, 20, 21, 22, 23, 24, 25, 26]. The provably near-optimal approach that we propose balances these two naive extremes to achieve both high quality and lower rebuffering.

Our Contributions. We leverage Lyapunov optimization techniques to achieve *both* high bitrates and low rebuffering by judiciously downloading higher-quality segments for tiles that are more likely to be in the FOV of the users, while using lower-quality segments for the rest of the tiles as a hedge against rebuffering. Our algorithm, BOLA360, is the first *provably* 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 formulate maximizing the quality-of-experience (QoE) of 360° videos as an optimization problem that we call ABR360. We model QoE as a weighted sum of two terms, one term relates to the quality (i.e., bitrate) of the video segments 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 uncertain 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 the performance of BOLA360 and show that (i) it never violates the buffer capacity of the client (Theorem 1), and (ii) its average QoE is within a small additive constant factor of the offline optimum of ABR360 (Theorem 2), the additive factor goes to zero when the buffer size goes to infinity. Further, let *playback delay* be the time elapsed from when a video segment is downloaded by the client to when it is rendered to the user. Our theoretical analysis reveals a tradeoff between playback delay and QoE of BOLA360, i.e., one needs to tolerate a longer playback delay to achieve better QoE (Remark 2).

4) We implement BOLA360 on a simulation testbed and evaluate its performance extensively using both real and synthetic data traces. Using trace-based simulations, we compare BOLA360 with both baseline and state-of-the-art algorithms used in VA-360 [27], 360ProbDASH [28], Salient-VR [29], Flare [30], and Pano [31]. Our results show that BOLA360 achieves a QoE that is better than the best

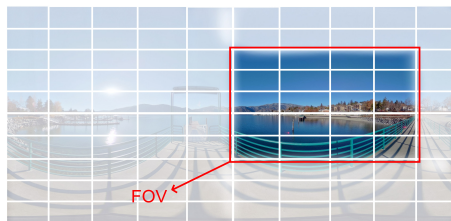


Figure 2: One shot from the entire spatial area of 360° video and FOV of user in that

alternative by average 13.6% over 14 real network profiles (Figure 6) and 30.3% over multiple videos and 12 different head position probability distributions (Figure 8).

5) Finally, we consider two extensions to BOLA360 that are relevant in specific real-world situations [32]. While BOLA360 exhibits impressive performance in terms of achieving high QoE, average playing bitrate, and rebuffering ratio, there is additional room to improve the QoE by adding heuristics on top of the basic design of BOLA360. Toward this, we improve BOLA360 by including minimizing fluctuations in the bitrate of rendered segments or ensuring swift responses to network conditions. To address these areas and enhance the practical performance of BOLA360, we propose two innovative heuristics: BOLA360-PL, and BOLA360-REP, each targeting specific drawbacks of the original algorithm. Our experimental results reveal substantial enhancements achieved by two heuristics. Specifically, BOLA360-PL reduces reaction time by up to 67.8%, and BOLA360-REP significantly improves both playing bitrate and reaction time by 91.2% and 80.0% respectively, especially when coupled with short-term head position predictions. These heuristics provide highly efficient and practical solutions, surpassing the performance of the original algorithm.

Roadmap. The rest of paper is organized as follows. We introduce our system model and formulate the ABR problem in Section 3. Using a Lyapunov optimization approach, we develop BOLA360 and prove that it is near-optimal in Section 4. We empirically analyze the behavior of BOLA360 in Section 5 and evaluate its performance in Section 6. Next, in Section 7, we introduce two additional versions of BOLA360 which practically improves the performance of BOLA360. Finally, we review related work in Section 8 and 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 segments to cover entire 360° spatial area. The user’s screen includes multiple tiles and each segment represents a short fraction of video for a particular tile. Usually, each segment is encoded in multiple quality levels or bitrates for video streaming. The ABR algorithm for 360° video has to select the bitrate of segment for each 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°[33]. 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 (say), the user can arbitrarily change the FOV and view by moving their head,

Navigation Graph. *Navigation Graph* for 360° videos introduced in [34] for the first time and is used to represent the probability of users transitioning from each view to another view as they watch the 360° video. Each node (k, v) of the graph corresponds to a k^{th} chunk of the video and view index v . Also, an weighted edge $e = \{k, v_i, v_j\}$ from node (k, v_i) to node $(k + 1, v_j)$ shows the probability of jumping from view v_i to view v_j while the chunk k is playing. Usually, the navigation graph is used

to keep the historical head direction traces of multiple users against a single video or historical head direction traces of a single user against multiple videos. Every time the user interacts with a video and jumps from view v_i to view v_j at chunk k the weight of edge $e = \{k, v_i, v_j\}$ get updated.

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 *segments* to cover the entire 360° spatial area. Each segment represents δ seconds of video for a particular tile of the screen. Moreover, each segment 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 of a segment with bitrate m . We define v_m as the utility value the user gets by watching a tile playing a segment with bitrate m . Therefore, we have the following inequality.

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

During the playback time of each chunk, the user views only a subset of tiles, which is their FOV. The bitrate of tiles inside the FOV directly impacts the QoE. On the other hand, downloading segments for tiles 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 the segment for 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 [35, 19, 36, 29, 34], or from a chunk analysis of the content [37, 38]. For simplicity, we assume that the FOV includes a single tile and $\sum_{d=1}^D p_{k,d} = 1$. The algorithm's design could be straightforwardly extended to include multiple tiles for the FOV of the user.

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 quality of experience (QoE) of the user, including two terms: 1) the utility term that is related to quality of the video watched by the user, such utility is an increasing function of the quality of the segment, 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 segments 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 segments. However, to maximize the expected continuous smooth playback, the ABR algorithm must download low-bitrate segments. Thus, to maximize the sum of both terms, the ABR algorithm must balance the two conflicting requirements.

We now formulate QoE mathematically captures the utility as the sum of the two terms U_K and R_K . U_K represents the time-average expected playback utility the video player prepares for the user over the sequence of segments and is defined as follows.

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

where T_{end} is the moment video player finishes playback time of the last chunk, and $a_{k,d,m}$ is a binary optimization variable in the ABR360 problem: $a_{k,d,m} = 1$ if segment with bitrate m is selected to download for tile d of chunk k ; 0, otherwise. Let t_k denotes the time the video player completes the

download of segments that belong to chunk $k - 1$ and decides about the segments of k^{th} chunk. And T_k shows the time interval between finishing downloading chunks $k - 1$ and k , i.e., $T_k = t_{k+1} - t_k$. In Equation (1), $p_{k,d}$ is the probability of the tile d being inside FOV during playback time of k^{th} chunk.

The second QoE term is denoted by R_K , which targets the playback smoothness as follows.

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

That is, R_K is the ratio of expected playback length of downloaded segments of video and the length of the streaming. Note that a low value for R_k when the the download time (denominator) is larger than the playback length of the segments (numerator) will result in rebuffering. Thus a large value of R_k is a measure of continuous play. In contrast to U_k , R_K has an inverse relation with the download time (or the bitrate), so it decreases with higher bitrates. Note that the expectations in Equation (1) and Equation (2) are over different possible decisions BOLA360 may take.

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.

$$\text{[ABR360]} \quad \max \quad U_K + \gamma R_K \quad (3a)$$

$$\text{s.t.}, \quad \sum_{m=1}^M a_{k,d,m} \leq 1, \quad \forall d, k, \quad (3b)$$

$$Q(t_k) \leq Q_{max}, \quad (3c)$$

$$\text{vars.}, \quad a_{k,d,m} \in \{0, 1\}. \quad (3d)$$

Constraint (3b) limits to downloading at most one segment for each tile of a chunk. The second constraint (3c) enforces the buffer capacity constraint. In this constraint, $Q(t_k)$ is the buffer level at time t_k and includes the number of segments available in a buffer at time t_k . Q_{max} is the buffer capacity and depicts the maximum number of segments stored in the buffer. Since the number of segments downloaded for each chunk is not fixed, the actual number of segments 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 segments downloaded for chunks played during downloading of chunk k . The evolution of the buffer level is characterized as

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

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

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

4 BOLA360: An Online ABR Algorithm for 360° Videos

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

Algorithm 1: BOLA360 (k)

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

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. 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 segments and reduce the input rate of the buffer. 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 proceed to explain the technical details of BOLA360. 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 segment k , the buffer level $Q(t_k)$ and head position probability values encoded in $p_{k,d}$ are given. BOLA360 selects the bitrates for segments of chunk k by solving the maximization problem described in the following.

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

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

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

where $\mathbf{a}(k)$ is a decision vector of the BOLA360 and

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

is a control parameter that is bounded the R.H.S term to guarantee the required buffer level for BOLA360 is less than Q_{max} . Constraint (5b) limits BOLA360 to download at most one segment for each tile. BOLA360 selects the near-optimal bitrates of chunk k by finding a decision vector $\mathbf{a}(k) = [a_{k,1,1}, a_{k,1,2}, \dots, a_{k,1,M}, a_{k,2,1}, \dots, a_{k,D,M}]$ that maximizes the value of $\eta(k, \mathbf{a}(k))$ in Equation (5a). When the buffer level exceeds $V(v_M + \gamma\delta)$, the algorithm decides to wait and download nothing (entering idle state). 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 [17], the algorithm waits until the buffer level reaches $Q(t_0) \leq V(v_M + \gamma\delta)$. We note that our theoretical analysis is valid even with a dynamic waiting time. The pseudocode for action taken by BOLA360 for segment k is described in Algorithm 1.

4.2 Theoretical Analysis of BOLA360

Our theoretical analysis first provides an upper bound for the buffer level while running BOLA360 in Theorem 1. Second, in Theorem 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 1. *Under bitrate control of BOLA360, the buffer level never exceeds $V(v_M + \gamma\delta) + D$,*

$$Q(t_k) \leq V(v_M + \gamma\delta) + D, \quad (6)$$

A proof of Theorem 1 is given in Appendix A.

Theorem 2. *Let OBJ be the QoE achieved by BOLA360. For a large video, i.e., $K \rightarrow \infty$,*

$$OBJ^* - \frac{D\delta^2 + \Psi}{2V\delta^2}\sigma \leq OBJ, \quad (7)$$

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

A proof of Theorem 2 is given in Appendix B. Our theoretical analysis assumes that the number of chunks is very large for the video, i.e., $K \rightarrow \infty$. Note that this assumption is needed for the theoretical analysis, and the algorithms do not need such an assumption.

Remark 2 (On the conflict between the playback delay and QoE of streaming). *Theorem 2 states as the value of V increases, the performance of BOLA360 gets closer to the optimal QoE. However, Theorem 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 Understanding the Behavior of BOLA360

To understand the detailed behavior of BOLA360, in this section, we evaluate the actions and performance of the BOLA360 using a trace-based simulation with synthetic traces. In the next section, we conduct a comprehensive study comparing BOLA360 with other state-of-the-art algorithms using both real and synthetic trace data.

5.1 Experimental Setup

We conducted our experiments using a video with a duration of 250 seconds, divided into chunks of 5 seconds each. Each chunk is further divided into six tiles, and each segment is encoded at six different bitrates. To represent utility values, we employed a logarithmic function, similar to previous works such as [39, 17, 40]. 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

Table 1: Available bitrates and utility values

Bitrate (Mbps)	0.2	0.4	0.6	0.8	1	1.5
Sizes (Mb)	1	2	3	4	5	7.5
Utility values	0.000	0.693	1.099	1.386	1.609	2.015

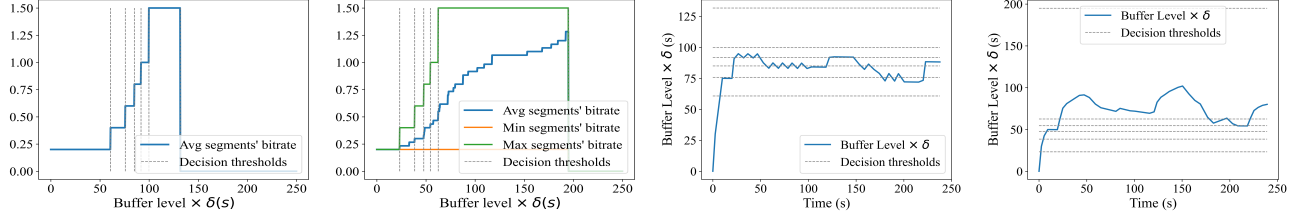


Figure 3: The selected bitrate of BOLA360 for tiles with highest and lowest probability and average selected bitrate as a function of buffer level for homogeneous (left most) and heterogeneous (second left) distributions, and buffer level variation over time for homogeneous (third left) and heterogeneous (right most) distributions. Marked threshold values show the buffer levels where the bitrate for tile with highest probability changes.

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 1, we present the utility values generated using the logarithmic function $v_m = \log(S_m/S_1)$. It is important to note that this utility function assigns a zero utility value to the lowest available bitrate. Although selecting the lowest bitrate does not affect the value of U_K , its positive impact on R_K makes it a better choice compared to not downloading any segment at all.

The head position of the user watching the 360° video is represented by a head position probability distribution that is critical for guiding the actions of BOLA360. In this section, 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 respectively. Additionally, we set the values of γ and V to be $\gamma = 0.1$ and $V = 1.66$ for the experiments conducted in this section. In this section, our goal is to understand the behavior of BOLA360 using these synthetic inputs, while we use realistic probabilities derived from real-world scenarios in the next section,

5.2 Experimental Results

Figure 3 shows the maximum, minimum, and average bitrates of segments downloaded by BOLA360 for each chunk of the video. For the homogeneous distribution, the selected bitrate for all segments of a chunk is the same. BOLA360 chooses its action by solving the maximization problem defined in Equation (5). This action is taken based on the current buffer level. 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 $V(v_M \cdot p_{k,d} + \gamma\delta)$, BOLA360 downloads nothing for that tile and tries to select the bitrate after Δ seconds. Note that increasing the value of γ increases the importance of continuous playback without rebuffers. Increasing the value of γ by ϵ is similar to reducing the buffer level by $\epsilon\delta V$, resulting in BOLA360 using correspondingly higher

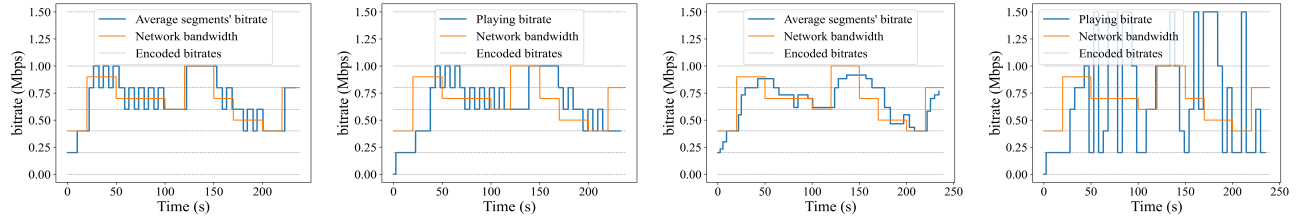


Figure 4: Variation of average downloaded bitrates and playing bitrate over time under bitrate selection of BOLA360 for the homogeneous (left most and second left), and heterogeneous (third left and right most) head position probability distribution. BOLA360 responds to the variation of network bandwidth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

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 segments downloaded across all tiles 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 of BOLA360 with other approaches

We introduce several algorithms that capture baseline techniques as well the current state-of-the-art known in the prior literature for solving ABR360. We provide an extensive comparison of the QoE achieved by BOLA360 in comparison with these algorithms and show that BOLA360 significantly advances the current state-of-the-art.

6.1 Comparison Algorithms

The first comparison algorithm, named DP_{on} , utilizes the estimated bandwidth to determine the bitrates of segments for the next chunk. However, DP_{on} lacks foresight and does not consider future implications, focusing solely on maximizing the immediate impact on the quality of experience (QoE). The second comparison algorithm, $Top-D$, distributes the estimated bandwidth equally among all D tiles and selects bitrates accordingly. $Top-D$ is very similar to the algorithm used in some previous works like [41, 28]. The third comparison algorithm, $VA-360$, is introduced in [27], which gives a unique weight to each tile and distributes the estimated bandwidth among all tiles based on the given weights, where the head movement probabilities serve as the weight of tiles. The fourth comparison algorithm, $360ProbDASH$ proposed in [28] selects the aggregate bitrate of tiles for each segment to keep the buffer level close to the targeted buffer level. Then, it distributes the selected bitrate among all tiles to maximize QoE and reduce the variance of tiles' bitrates inside the FOV. The last comparison algorithm, $Salient-VR$, proposed in [29]. $Salient-VR$, leverage the estimated bandwidth and buffer level to determine the highest possible bitrates such that the download time of a chunk does not exceed the length of the buffered video. Note that there are other state-of-the-art algorithms, such as Flare [30], or Pano [31], which consider different metrics, like minimizing bitrate variations across segments of a chunk. However, these algorithms may exhibit weaker performance when evaluated using the QoE defined in this work. By adapting their concepts to align with the QoE defined in this work, we can develop ABR360 algorithms that closely resemble $360ProbDASH$, DP_{on} , or $Top-D$.

Table 2: Available bitrates and utility values of them for experiments of Sections 6.3, and 6.4

bitrate (Mbps)	0.44	0.7	1.35	2.14	4.1	8.2	16.5
Sizes (Mb)	2.2	3.5	6.75	10.7	20.5	41.0	82.5
Utility values	0.000	0.464	1.121	1.582	2.232	2.925	3.624

6.2 Experimental Setup

To evaluate the performance of these algorithms, we conducted experiments in multiple different scenarios to demonstrate the algorithms’ performance under different settings. We use a 250 seconds video, split into chunks of 5 seconds and spatially distributed over 8 tiles. Also, the video is encoded in seven different bitrates, i.e., $M = 7$. Similar to Section 5, we use the logarithmic utility function. The list of available bitrates, size of segments, and utility values are listed in Table 2. The buffer capacity is $Q_{\max} = 64$. Finally, we select $\gamma = 0.3$, $V = 10.9$, and dynamic value selection for Δ as suggested in [17]. We use 4G bandwidth traces from [42] and 4G/LTE bandwidth trace dataset [43] collected by IDLAB [44] to simulate the network condition. We select 14 different traces from 4G/LTE dataset to evaluate the performance of BOLA360 under different network conditions. During our evaluation process, the video is stored on an Apache server. Both server and client use Microsoft Windows as a OS, 24GB of RAM, and 8-core, 3Ghz Intel Core-i7 CPU. We used Chrome DevTools API [45] to transfer the video between server and client and also emulate the network condition. We fetched the bandwidth capacity from the 4G/LTE dataset and injected them into the Chrome DevTools to limit the download capacity between the server and the client. Unless anything else mentioned, to capture the actual FOV of the user and head position probability values, we generate the navigation graph [34] for 360° video using public VR head traces published by [46].

6.3 Performance Evaluation using Real Network and Head Movement Traces

In this experiment, we compare the performance of BOLA360 and other comparison algorithms using real network and head movement traces. We use 4G bandwidth traces, network profile 15 in Appendix D for this section. We report playing bitrate, the rebuffering ratio (percentage of length of video considered as a rebuffering), and QoE of BOLA360, DP_{on}, Top-D, VA-360, and Salient-VR. Note that, the average playing bitrate reported in Figure 5 is calculated over the segments 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, video, and algorithm parameters. The CDF plot of average bitrates, rebuffering ratio, and QoE values of 100 different trials is reported in Figure 5.

The results in Figure 5 show that BOLA360 substantially outperforms other comparison algorithms in QoE as well as the bitrate of a tile user has seen. VA-360 selects relatively high bitrates for all segments of a chunk while BOLA360 efficiently distributes the available bitrates among different segments such that BOLA360 is able to achieve higher playing bitrate and also lower rebuffering ratio. It is worth noting that the rebuffering of DP_{on} is very low since it is designed to minimize the rebuffering by taking actions with an expected download time of less than δ seconds regardless of buffer level. However, its average bitrate is significantly lower than that of BOLA360.

Key takeaway. BOLA360 outperforms comparison algorithms in terms of QoE as it is designed to maximize it. Besides, BOLA360 performs better than *all* comparison algorithms in terms of playing bitrate and rebuffering ratio.

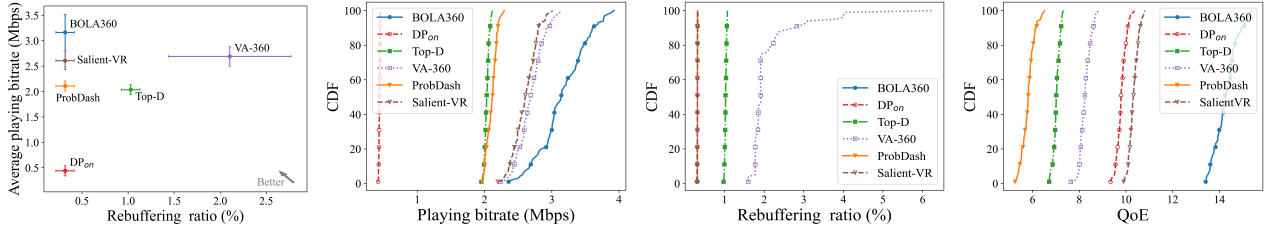


Figure 5: Average playing bitrate vs. rebuffering ratio (left most), the CDF of playing bitrate (second left), rebuffering ratio (third left), and QoE (right most) of BOLA360 and comparison algorithms using real network and head movement traces. The average playing bitrate of BOLA360 was $3.2Mbps$, while this value for VA-360, Salient-VR, and 360ProbDASH were $2.6Mbps$, $2.6Mbps$, and $2.1Mbps$. However, the average rebuffering for BOLA360, VA-360, Salient-VR, and 360ProbDASH were 0.26% , 2.1% , 0.25% and 0.25% , respectively.

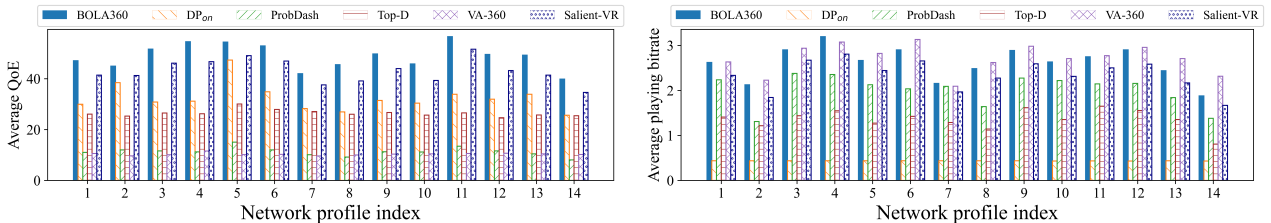


Figure 6: The average QoE (left) and average playing bitrate (right) over the bitrate selection of BOLA360 and other comparison algorithms for 14 different network profiles and 100 trials. In terms of QoE, BOLA360 outperforms others in all profiles. On average, BOLA360 achieves about 13.6% more QoE than Salient-VR which was in the second place during these experiments.

6.4 Impact of Network Bandwidth on the Performance of Algorithms

The bandwidth capacity and variations significantly impact the performance of online algorithms for ABR360. In this experiment, we investigate the impact of different network profiles on the performance of BOLA360 and comparison algorithms.

We use 14 different network traces from 4G/LTE dataset [43] to generate the bandwidth throughput, which are all shown in Appendix D. We use the same video and algorithm/problem parameters (details in Section 6.2) for all algorithms to capture the impact of the network capacity on their performance. The results are reported for 100 trials for each network profile. We report the average QoE, playback delay, rebuffering ratio, and average playing bitrate of algorithms.

The results in Figure 6 report the average QoE, 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 these experiments, VA-360 selects relatively higher bitrates compared to other algorithms, while its high rebuffering, shown in Figure 7 is substantial such that it lowers the QoE of this algorithm as compared to BOLA360. In addition the playback delay of BOLA360 and comparison algorithms shown in Figure 7. We can see 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 less than 7.8 seconds while its rebuffering ratio was the lowest among comparing algorithms in 12 out of 14 experiments.

Key takeaway. Networks with high fluctuations (e.g., profile index 2 and 7) causes a high rebuffering ratio and leads BOLA360 to select bitrates with more cautious. Results in lower playing bitrates (compared to other network profiles).

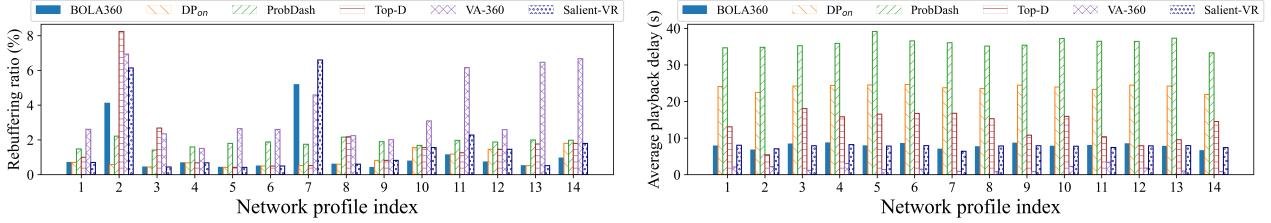


Figure 7: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of BOLA360 and comparison algorithms for 14 different network profiles and 100 trials. VA-360 usually results in higher rebuffering compared to the other algorithms while its playback delay is very short. The average playback delay for BOLA360 was 7.8 seconds.

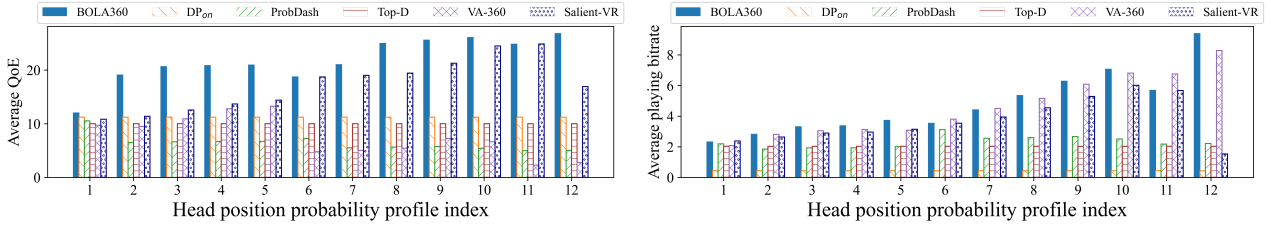


Figure 8: The average QoE (left) and average playing bitrate (right) over the bitrate selection of BOLA360 and comparison algorithms using 12 different head position probability distributions over 100 trials. On average, BOLA360 achieves about 30.3% more QoE than Salient-VR which was the second best algorithm in 11 out of 14 experiments.

6.5 Impact of Head Position Probabilities on the Performance of Algorithms

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 probability distribution on the performance of ABR algorithms, we define 12 probability distributions (details in Appendix C). We evaluate the BOLA360 and comparison algorithms against these 12 probability distributions while the rest of the setting is similar to the experiment in Section 6.3. Specifically, for each chunk k , we replace the set of probabilities with the probabilities calculated from Equation (14) in Appendix C. Note that each head position probability distribution could be interpreted as a different video file. 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 relatively higher QoE when the prediction of FOV is concentrated on fewer number of tiles. Since the Top-D downloads the fixed bitrate for all tiles, the expected QoE of Top-D is independent of the head position probability distribution. A notable observation from playing bitrate depicted in Figure 8 demonstrates that BOLA360 kept the average playing bitrate at a high value for every probability profile, while the achieved QoE is promising, and kept rebuffering ratio close to the lowest among all algorithms.

Key takeaway. The playing bitrate of BOLA360 and most comparison algorithms improves when the head position prediction is concentrated on fewer number of tiles. Meanwhile, BOLA360 could improve the playing bitrate more than other comparison algorithms.

6.6 Discussion on the Performance of Predictions-based Algorithms

This section provides details on why the baseline or state-of-the-art algorithm used in Section 6.2 may fail to perform well in particular scenarios, and they cannot guarantee their performance in the worst-case scenario. All of Top-D, DP_{on}, VA-360, 360ProbDASH, and Salient-VR algorithms take

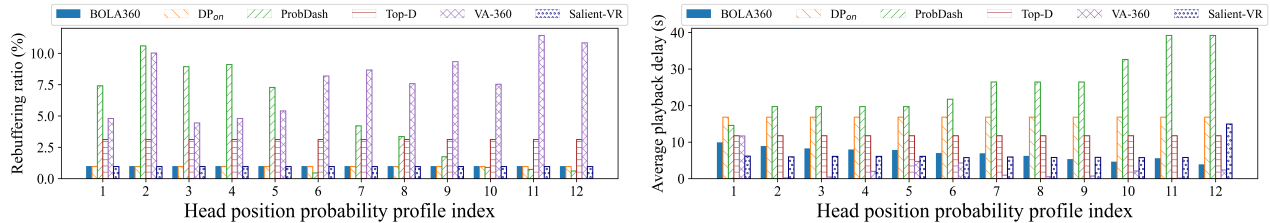


Figure 9: The average rebuffering ratio (left) and average playback delay (right) over the bitrate selection of BOLA360 and comparison algorithms using 12 different head position probability distributions over 100 trials. VA-360 usually results in high rebuffering ratio and short playback delay; meanwhile, the rebuffering ratio of BOLA360 was the lowest in 8 out of 12 experiments. The average playback delay for BOLA360 was 6.7 seconds.

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, Top-D, DP_{on} 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 segment upgrade into the BOLA360. The basic BOLA360 algorithm is not designed to replace previously downloaded segments with higher bitrates, further restricting its adaptability. Consequently, if the network condition momentarily deteriorates and subsequently improves, the algorithm toggles between lower and higher bitrates. While a high buffer level grants ABR algorithm an opportunity to replace low bitrate segments downloaded earlier, the fundamental design of BOLA360 fails to support this crucial action. A detailed explanation of both heuristics is given below.

BOLA360-PL is a generalized version of BOLA-PL introduced in [32]. It aims to reduce the reaction time of the BOLA360 during start and seek times. The reaction time is the duration from when the first segment is fetched (during start time) or the first seek segment is fetched (during seek time) until the selected bitrate of tiles by BOLA360 stabilizes. The main concept behind BOLA360-PL is to virtually increase the buffer level at the start or seek time. This is achieved by estimating the bandwidth and multiplying it by 50% to establish a safe expected bandwidth. To prevent rebuffering, BOLA360-PL limits the bitrate of each segment 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 expected bandwidth capacity at time t . BOLA360-PL virtually inserts a proportional number of segments into the buffer such that the size of the new downloading chunk does not exceed S_{lim} .

The second heuristic is BOLA360-REP, which is a variant of BOLA360 that allows for upgrading of previously downloaded segments. One limitation of BOLA360 is its inability to modify previously

downloaded segments. Specifically, BOLA360 must make decisions about the next chunk, and it is not designed to replace higher bitrate segments with previously downloaded, lower-quality ones. BOLA360-REP determines whether it is better to download a new segment for the next chunk or to improve the quality of previously downloaded segments based on the length of video available in the buffer. If the decision is to download segments for the next chunk, BOLA360-REP selects the bitrates according to the decision of BOLA360. If the decision is to replace the previously downloaded segments, BOLA360-REP identifies a tile where there is at least a two-level difference between the bitrate of downloaded segment for that tile and the bitrate that BOLA360 would select for that tile at the current time. BOLA360-REP then downloads and replaces the new segments for those low-quality segments.

7.1 Experimental Setup

We choose the parameters used in Section 6.5 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 $p_{k,d} = 1/D$.

7.2 Experimental Results

Figure 10 shows the CDF plots of the average segments’ bitrate (left), the reaction time (middle), and oscillation (right, the average difference between the bitrate of two consecutive segments) 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 segments during start and seek time, and these segments 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 segments’ 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%. In addition, BOLA360-REP 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.

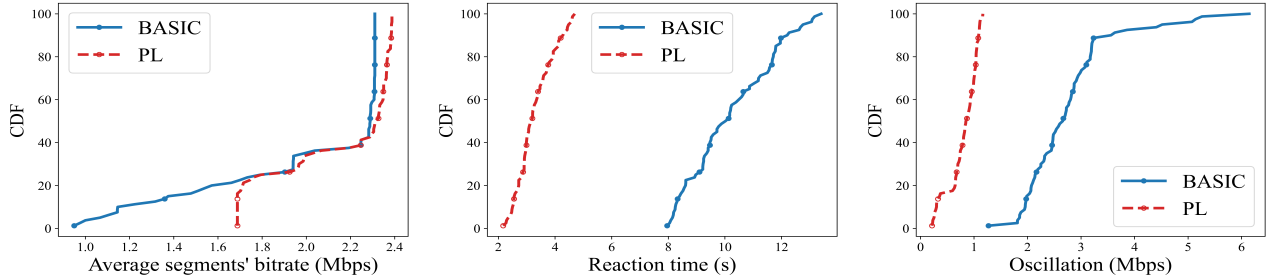


Figure 10: The CDF of the average bitrate of any downloaded tile (left), reaction time (middle), and oscillation (right) of basic BOLA360 and BOLA360-PL using real network and head movement traces. BOLA360-PL reduces the oscillation and reaction time by 70.9% and 67.8% respectively.

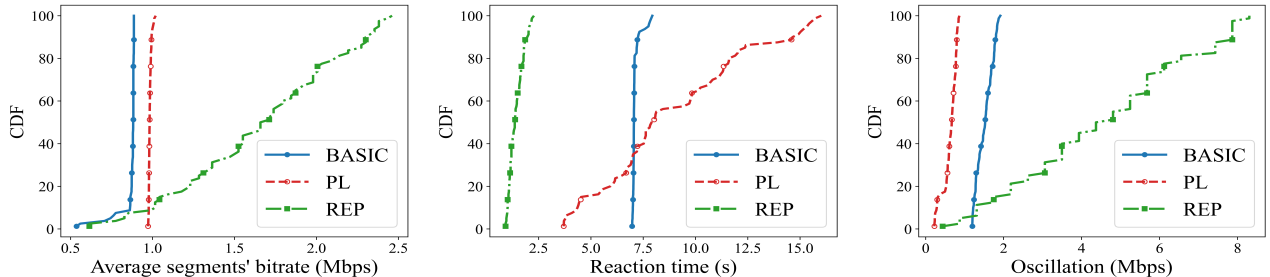


Figure 11: The CDF of average bitrate of downloaded segments (left), reaction time (middle), and oscillation (right) of basic BOLA360, BOLA360-PL, and BOLA360-REP using real network and head movement traces while the prediction of the head position dynamically got updated. BOLA360-REP improves the average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

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 incorporate them into existing ABR algorithms. For example, [30] 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 [47] propose a Lyapunov-based model that uses Lyapunov optimization to solve ABR360 problem. In their work, the quality of the tiles is selected based on the motion maps. In addition, they add saliency map information to their model to make a balance between QoE and playback delay. In another work, [48] 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 update the tiles inside the short buffer layer (short-term improvement). The long buffer layer tries to download the new segments that are not available in the short buffer layer and will be played later. In another work, [41] 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 [49, 50, 51, 52, 18], several deep RL-based algorithms are developed for solving bitrate selection problems. 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 [53], 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 [54, 29, 38, 55, 28] 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. [31] employ Model Predictive Control (MPC) to select the aggregate bitrate for a segment, allocating it among tiles to maintain quality within the limited bitrate. In another category of research [56, 57], 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 DP_{on} (Section 6).

9 Conclusion and Future Directions

In this paper, we formulated an optimization problem to maximize users’ quality of experience in 360° video streaming applications. Then, 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 that maximizes the quality while ensuring the rebuffering rate is minimal. Our comprehensive experimental results showed that BOLA360 performs better than several other alternative algorithms under a wide range of network and head movement profiles. In future work, we plan to develop a data-driven and robust version of BOLA360 to explicitly use the future predictions in the decision-making while persevering the theoretical performance guarantees of the algorithm.

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: 2022-03-01.
- [2] Sony. Sony playstaion vr. <https://www.playstation.com/en-us/ps-vr2/>, 2022. Accessed: 2022-03-01.
- [3] Microsoft. Microsoft vr headset. <https://www.microsoft.com/en-us/store/b/virtualreality>, 2022. Accessed: 2022-03-01.
- [4] Google LLC. Google ar/vr. <https://arvr.google.com/ar/>, 2022. Accessed: 2022-03-01.
- [5] 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: 2022-08-01.
- [6] 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.
- [7] 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.

- [8] 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.
- [9] Malleshham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1977–1986. IEEE, 2020.
- [10] Akamai. Akamai’s [state of the internet]. <https://www.akamai.com/site/en/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-infographic.pdf>, 2017. Accessed: 2022-07-18.
- [11] EtiSoftware. Internet speed and subscriber dissatisfaction. <https://tinyurl.com/network-speed>, 2021. Accessed: 2022-07-18.
- [12] 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.
- [13] Bo Han, Yu Liu, and Feng Qian. Vivo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–13, 2020.
- [14] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.
- [15] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.
- [16] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. Sensei: Aligning video streaming quality with dynamic user sensitivity. In *NSDI*, pages 303–320, 2021.
- [17] Kevin Spiteri, Rahul Uргаonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.
- [18] 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 the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.
- [19] Chenge Li, Weixi Zhang, Yong Liu, and Yao Wang. Very long term field of view prediction for 360-degree video streaming. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (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 Workshop on Network and Operating Systems Support for Digital Audio and Video*, 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 *2018 IEEE International Conference on Multimedia and Expo (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 *2018 IEEE International Conference on Multimedia & Expo Workshops (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 *2020 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 Multimedia Systems Conference*, pages 132–145, 2021.
- [27] Cagri Ozcinar, Ana De Abreu, and Aljosa Smolic. Viewport-aware adaptive 360 video streaming using tiles for virtual reality. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2174–2178. IEEE, 2017.
- [28] 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 international conference on Multimedia*, pages 315–323, 2017.
- [29] 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 the 28th Annual International Conference on Mobile Computing And Networking*, pages 542–555, 2022.
- [30] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114, 2018.
- [31] 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 Special Interest Group on Data Communication*, pages 394–407. 2019.
- [32] 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 (TOMM)*, 15(2s):1–29, 2019.
- [33] 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.
- [34] Jounsup Park and Klara Nahrstedt. Navigation graph for tiled media streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 447–455, 2019.
- [35] 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.

- [36] 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.
- [37] 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.
- [38] 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 international conference on Multimedia*, pages 564–572, 2018.
- [39] 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.
- [40] Han Hu, Cheng Zhan, Jianping An, and Yonggang Wen. Optimization for http adaptive video streaming in uav-enabled relaying system. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.
- [41] Sohee Park, Arani Bhattacharya, Zhibo Yang, Mallesh Dasari, Samir R Das, and Dimitris Samaras. Advancing user quality of experience in 360-degree video streaming. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.
- [42] A Bokani, M Hassan, SS Kanhere, J Yao, and G Zhong. Com-prehensive mobile bandwidth traces from vehicular networks. In *Proceedings of the 7th International Conference on Multimedia Systems. Association for Computing Machinery*, 2016.
- [43] Ghent University. 4g/lte bandwidth logs. <https://users.ugent.be/~jvdrhoof/dataset-4g/>, 2019. Accessed: 2022-06-20.
- [44] 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.
- [45] Google LLC. Google-chrome: Chrome devtools protocol. <https://chromedevtools.github.io/devtools-protocol/tot/Network/>, 2023. Accessed: 2023-01-21.
- [46] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A dataset for exploring user behaviors in vr spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 193–198, 2017.
- [47] Wang Shen, Lianghai Ding, Guangtao Zhai, Ying Cui, and Zhiyong Gao. A qoe-oriented saliency-aware approach for 360-degree video transmission. In *2019 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2019.
- [48] 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.
- [49] 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 2019-IEEE Conference on Computer Communications*, pages 1252–1260. IEEE, 2019.

- [50] 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.
- [51] Jun Fu, Chen Hou, and Zhibo Chen. 360hrl: Hierarchical reinforcement learning based rate adaptation for 360-degree video streaming. In *2021 International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–5. IEEE, 2021.
- [52] 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.
- [53] 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.
- [54] 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 international conference on Multimedia*, pages 1190–1198, 2018.
- [55] 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 the 25th ACM international conference on Multimedia*, pages 306–314, 2017.
- [56] 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 Annual International Conference on Mobile Systems, Applications, and Services*, pages 482–494, 2018.
- [57] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.
- [58] Michael J Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.
- [59] Michael J Neely. Dynamic optimization and learning for renewal systems. *IEEE Transactions on Automatic Control*, 58(1):32–46, 2012.

A More Detail on the Performance of BOLA360

Remark 3. *Theorem 1 gives an upper bound for the buffer level used by BOLA360 as a function of problem parameters and algorithm parameter V . The buffer usage of BOLA360 grows linearly with the growth of V . Buffer level usage is an upper bound for the playback delay of the video. Based on this theorem, one can target lower playback delay decreasing the value of V .*

We prove this theorem by induction. The base case is $Q(t_1) = 0 \leq V(v_M + \gamma\delta) + D$ that satisfies the statement of the theorem. Two cases are possible for value of $Q(t_k)$:

- *Case 1:* $Q(t_k) \leq V(v_M + \gamma\delta)$: in this case, the buffer level at time t_{k+1} does not exceeds $Q(t_k) + D \leq V(v_M + \gamma\delta) + D$.
- *Case 2:* $V(v_M + \gamma\delta) < Q(t_k) < V(v_M + \gamma\delta) + D$: In this case, the action at time t_k is to download nothing for any tiles. Hence $Q(t_{k+1}) \leq Q(t_k)$.

Now, we proceed to analyze the QoE performance of BOLA360. With large K , the ABR360 problem with rate stability constraint [58] is equivalent to the relaxed version of ABR360 with limited buffer capacity, i.e.,

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

Any solution to ABR360 with limited buffer capacity requires that the expected input rate into the buffer be less than or equal to the buffer's output rate. Otherwise, for the case of $K \rightarrow \infty$, the limited buffer capacity constraint does not hold. It is good to mention that any solution to ABR360 with limited buffer capacity satisfies the rate stability constraint; however, the opposite statement is not necessarily true.

The stationary algorithm: In the context of ABR360 problem, we define *stationary algorithm* as an ABR algorithm that downloads a fixed set of bitrates for all segments. This algorithm uses a fixed set of bitrates, \mathbf{A}^* , with size D ($|\mathbf{A}^*| = D$), and for each chunk K , it selects the bitrates such that 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 [59]. Precisely, by fixing renewal frame duration and letting the achieved QoE of downloading each tile represent penalty values in the notation of [59], the offline ABR360 problem can convert into an optimization problem over renewal frames. Then, following Lemma 1 in [59], we prove the existence of a stationary algorithm with QoE of $U_K^* + \gamma R_K^*$.

Lemma 3. *For the ABR360 problem with a large video, i.e., $K \rightarrow \infty$, there exists a stationary algorithm that satisfies the rate stability constraint and achieves the expected QoE of $U_K^* + \gamma R_K^*$.*

Proof. The proof of this lemma follows from Lemma 1 in [59] and continues with the approach taken for proof of Lemma 1 in [17]. 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 [59]. \square

B Proof of Theorem 2

Lets define the Lyapunov function $L(Q(t_k))$, and per-slot conditional Lyapunov drift $\Phi(t_k)$ as below

$$L(Q(t_k)) = \frac{1}{2}Q^2(t_k),$$

$$\Phi(t_k) = \mathbb{E}\{\Delta L(Q(t_k)) \mid Q(t_k)\} = \mathbb{E}\{L(Q(t_{k+1})) - L(Q(t_k)) \mid Q(t_k)\}.$$

Depending on the buffer level, the value of $\Phi(t_k)$ can get derived using buffer level evolution described in Equation (4). Consider two cases: 1) $Q(t_k) \leq n_k T_k / \delta$ and, 2) $Q(t_k) > n_k T_k / \delta$. In the first case, the value of $\Phi(t_k)$ would be

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

and in the second case the value of $\Phi(t_k)$ would be

$$\Phi_2(t_k) = \mathbb{E}\left\{\frac{1}{2}\left(\sum_{d=1}^D \sum_{m=1}^M a_{k,d,m} - \frac{n_k T_k}{\delta}\right)^2 - Q(t_k)\left(\frac{n_k T_k}{\delta} - \sum_{d=1}^D \sum_{m=1}^M a_{k,d,m}\right) \mid Q(t_k)\right\}.$$

Therefore, there would be an upper bound for the value $\Phi(t_k)$ of as below

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

Now, by subtracting $V\mathbb{E}\left\{\sum_{d=1}^D \sum_{m=1}^M a_{k,d,m}(p_{k,d}v_m + \gamma\delta) \mid Q(t_k)\right\}$ from both side, we have

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

Let w_{avg} denotes the average bandwidth capacity during downloading chunk k . Then the conditional expected download time for chunk k , $\mathbb{E}\{T_k \mid Q(t_k)\} = \mathbb{E}\left\{\frac{1}{w_{\text{avg}}} \sum_{d=1}^D \sum_{m=1}^M a_{k,d,m} S_m \mid Q(t_k)\right\}$, can get minimized without requiring any knowledge of w_{avg} as follows.

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

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

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

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 (8). T_k^* shows the length of download time for chunk k while the stationary algorithm is taking action. Based on rate stability constraint (8), the second term in equation above is always negative, so

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

By taking the sum over $k \in \{1, 2, \dots, K\}$ we have

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

By dividing all terms by $V \times K \times \mathbb{E}\{T_k\}$, we get

$$\frac{\mathbb{E}\{L(Q(t_K))\}}{VK\mathbb{E}\{T_k\}} - \text{OBJ} \leq \frac{D\delta^2 + \Psi}{2V\delta^2} \cdot \frac{1}{\mathbb{E}\{T_k\}} - (U_K^* + \gamma R_K^*).$$

Last, by taking the limit $K \rightarrow +\infty$ we get

$$(U_K^* + \gamma R_K^*) - \frac{D\delta^2 + \Psi}{2V\delta^2}\sigma \leq \text{OBJ}.$$

C Generating Probability Distributions for Head Position Probability Values Used in Section 6.4

To investigate the impact of head position probability values on the performance of different algorithms, we use synthetic probabilities to simulate different real-world scenarios. Toward this, we generate the head position probability distributions based on three parameters $D_{\text{pos}}(k)$, $r_{\text{min}}(k)$, and $\alpha_p(k)$. Parameter $D_{\text{pos}}(k)$ is the number of tiles with positive probability inside the head position probability values for chunk k , i.e., $D_{\text{pos}}(k) = |\{p_{k,d}|p_{k,d} > 0\}|$; $r_{\text{min}}(k)$ 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 introduce several additional auxiliary notations to concretely define $\alpha_p(k)$. Let $p_i^{(L)}(k)$ be the probability of i^{th} highest probable tile if there is a fixed step, Δ_p , between the probability of each tile and the probability of the next tile in sorted order of probabilities. We define the probability of i^{th} highest probable tile as a function of $\alpha_p(k)$ as follows.

$$p_i(k) = \frac{1 - \alpha_p(k)}{D_{\text{pos}}(k)} + \alpha_p(k)p_i^{(L)}(k). \quad (14)$$

The higher values of $\alpha_p(k)$, the higher the accuracy of the prediction of FOV. Note that we have $0 < D_{\text{pos}}(k) \leq D$, $r_{\text{min}}(k) \leq 1$ and $0 < \alpha_p(k) \leq 1$. With the above definition in Equation (14), $\alpha_p(k)$

Table 3: The details of the probability distributions used in the experiment of Section 6.5

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

determines the range of probabilities. In particular, $\alpha_p(k) = 0$ means the probability of all tiles are the same, i.e., $p_i(k) = 1/D_{\text{pos}}(k)$. On the other hand, $\alpha_p(k) = 1$ spans a wide range of probabilities between the minimum and the maximum., representing heterogeneous values for the 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 $r_{\text{min}}(k) = 0.05$ for all distributions. Also, the details of the 12 probability distributions used in Section 6.4 is listed in Table 3.

D Network Profiles Used In Section 6 and 6.4

The network profiles used in Section 6 and 6.4 are plotted in Figure 12. The network profile index 15 shows 4G bandwidth traces published in [42] and network profile 1-14 are from 4G/LTE bandwidth trace dataset [43] collected by IDLAB [44].

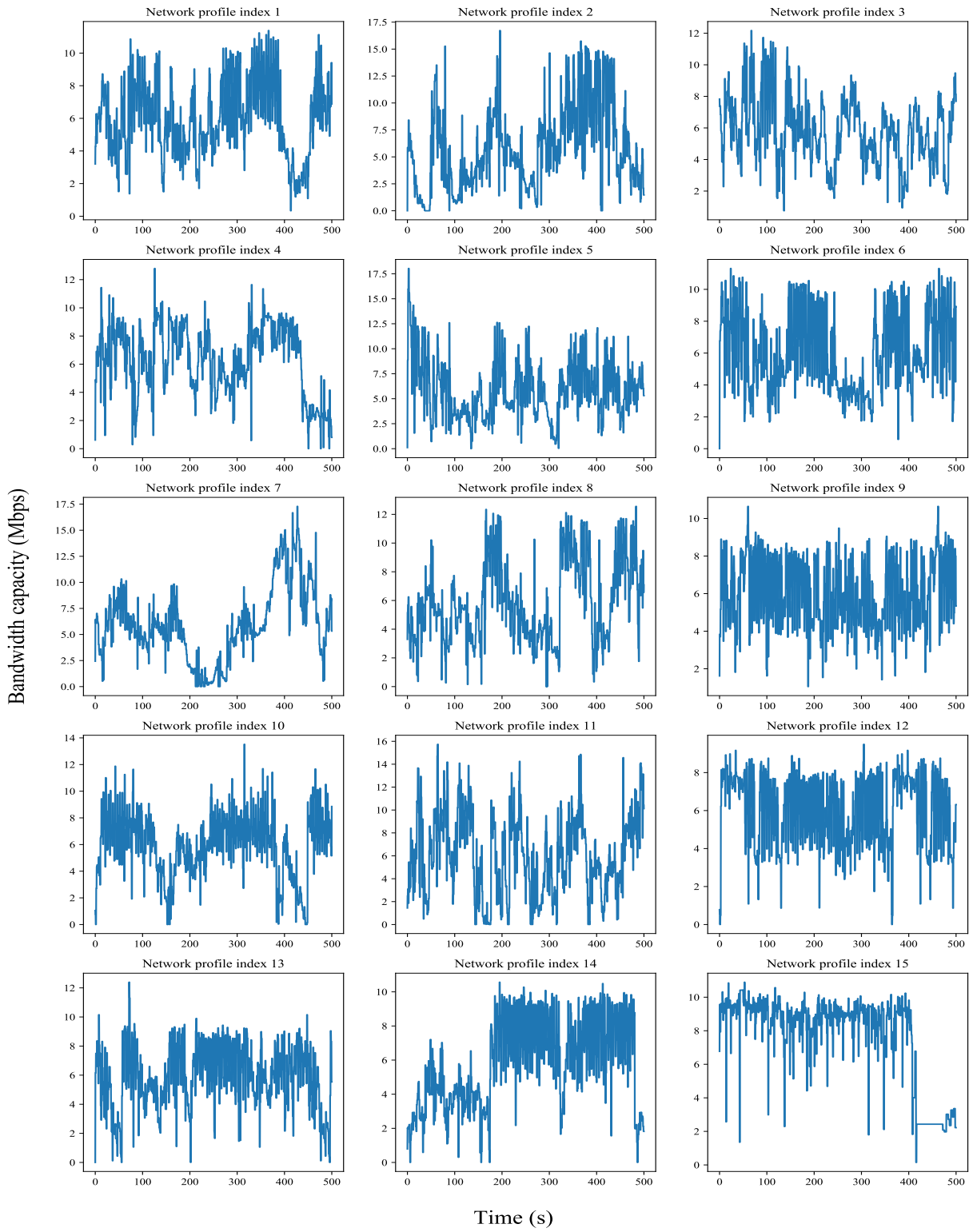


Figure 12: Fifteen network profiles used in experiments of Section 6 and Section 6.4.