Dragonfly: Higher Perceptual Quality For Continuous 360° Video Playback

Ehab Ghabashneh
Purdue University

Chandan Bothra
Purdue University

Ramesh Govindan
University of Southern California

Antonio Ortega
University of Southern California

Sanjay Rao
Purdue University

ABSTRACT

When streaming 360° video, it is possible to reduce bandwidth by 5× with approaches that spatially segment video into tiles and only stream the user's viewport. Unfortunately, it is difficult to accurately predict a user's viewport even 2-3 seconds before playback. This results in rebuffering events owing to misprediction of a user's viewport or network bandwidth dips, which hurts interactive experience. However, avoiding rebuffering by naively skipping tiles that do not arrive by the playback deadline may lead to incomplete viewports and degraded experience.

In this paper, we describe Dragonfly, a new 360° system that preserves interactive experience by avoiding playback stalls while maintaining high perceptual quality. Dragonfly prudently skips tiles using a model that defines an overall utility function to decide which tiles to fetch, and at which qualities they should be fetched, with the goal of optimizing user experience. To minimize incomplete viewports, it also fetches a low quality masking stream. Using a user study with 26 users and emulation-based experiments we show that Dragonfly has higher quality, and lower overheads, than state-of-the-art 360° streaming approaches. For instance, in our study, 65% of sessions have a rating of 4 or higher (Good/Excellent) with Dragonfly, while only 16% of sessions with Pano, and 13% of sessions with Flare achieve this rating.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Information systems** → **Multimedia streaming**;

KEYWORDS

360° Videos, Video Streaming, Adaptive Bit-Rate (ABR) Algorithms

ACM Reference Format:
Ehab Ghabashneh, Chandan Bothra, Ramesh Govindan, Antonio Ortega, and Sanjay Rao. 2023. Dragonfly: Higher Perceptual Quality For Continuous 360° Video Playback. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY , USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3603269.3604876

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604876

1 INTRODUCTION

The tremendous success of Internet video has led to a growing interest in new forms of video content in recent years. An increasingly popular new form, 360° video, is generated using 360° cameras, and permits viewers to choose any viewing angle within a 360° field of view [10, 17]. With 360° video users do not just passively consume content, but *actively* engage with it. Rather than view video in a linear fashion, and from publisher-specified perspectives, users may interactively traverse the content along many different paths from a perspective of their choice, with different users observing different perspectives of the same content.

Although in its infancy, there is much interest in 360° video in various domains including entertainment, education, and e-commerce [1, 3, 4, 17]. Recent surveys of video marketers and consumers [2, 15] indicate 360° video can lead to better user engagement and persuasion metrics.

Unlike traditional video, which must primarily handle uncertainty in network bandwidth (*network uncertainty*), 360° video must also deal with uncertainty in how the user interacts with the video (*motion uncertainty*). Two classes of solutions have emerged for 360° video. The first, used by traditional video publishers (e.g., YouTube, Facebook), treats 360° video like conventional video streaming, encoding and streaming the entire 360° view to the client. Unfortunately, while this permits users to view the content from any perspective, it is prohibitively expensive — it can consume 5-6× the bandwidth needed to stream only the relevant portions of the video to the user [7, 16, 29, 32].

A second class of approaches, which we term *view-centric*, involves *predicting* a user's *viewport* (portion of the video visible to the user), and tailoring transmission based on the prediction. These approaches stream the entire viewport [37, 38], or a subset [21, 24], at a higher quality. They transmit regions outside the viewport at lower quality [21, 24], or not at all [38]. Beyond temporally splitting video into chunks, as in conventional video streaming, they spatially segment each chunk into *tiles* (§2). Each tile may be independently encoded and downloaded, and may be encoded at different qualities. Most view-centric approaches stall playback if any viewport tile is unavailable prior to the playback deadline, and focus on optimizing perceptual quality of the viewport and minimizing stalls, much like traditional video streaming.

Challenge. View-centric approaches can significantly reduce bandwidth consumption as long as they correctly predict the viewport.

Unfortunately, the accuracy in predicting a user's viewport degrades significantly with look-ahead (i.e., how much in advance of playback the prediction is made). The accuracy can be as low as 25.5% (median) for a look-ahead of 3 seconds (§2). This makes it difficult to correctly handle network and motion uncertainty simultaneously. On the one hand, a small look-ahead window improves viewport prediction, but makes the scheme vulnerable to network bandwidth dips, potentially resulting in *network-induced stalls* in video playback. On the other hand, a large look-ahead window may result in *motion-induced stalls* since not all tiles relevant to the user's viewport may be fetched before playback. In fact, state-of-the-art systems [24, 38] report significant rebuffering.

Contributions. Existing approaches stall playback until *all tiles* relevant to a user arrive. This is especially undesirable in 360° settings, where a user may move during the stall event, potentially changing the viewport that must be fetched, resulting in further cascaded stalls (§2). Dragonfly, our proposed 360° streaming system, is explicitly designed for continuous playback.

To ensure continuous playback without stalls, a naïve approach would simply skip viewport tiles that do not arrive by the playback deadline. However, this may result in incomplete viewports with blank areas and unacceptable user experience. To avoid incomplete viewports. Dragonfly focuses on a design point that involves transmission of two streams: (i) a *primary* stream which encodes viewport tiles at high quality; and (ii) a lower quality *masking* stream to mask the effect of missing tiles.

While using two streams helps, it is challenging to ensure all viewport tiles are fetched in the primary stream by the rendering deadline. In fact, the problem is further exacerbated since some of the bandwidth is consumed by the masking stream. A direct adaptation of existing techniques [24, 38] would involve fetching all viewport tiles in the primary stream in as high a quality as possible before the rendering deadline, and simply skipping those that do not arrive by deadline (a *Passive Skip* strategy).

Dragonfly's central contribution is **proactive skipping**, in which some viewport tiles in the primary stream can be *selected* to be skipped. This is motivated by two observations. First, users may tolerate occasionally degraded (or even missing) content for some parts of the viewport (e.g., at the periphery). Second, proactively skipping tile fetches provides additional degrees of freedom that can be exploited to enhance user experience. For instance, it may be desirable to skip a tile with a more immediate deadline that only benefits a small number of frames and is at the viewport periphery, and instead fetch with higher quality a tile that is needed later but lies in the center of the viewport for several frames.

To achieve proactive skipping, Dragonfly computes a utility function for each tile, taking into account (i) the number of frames that could benefit if the tile were fetched by a given time; (ii) how central within the viewport the tile is to each frame; and (iii) the marginal perceptual gain of fetching that tile in higher quality. Guided by this utility function, Dragonfly schedules which tiles to fetch (or skip), in what order, and at what quality. Unlike existing scheduling algorithms used in systems that fetch all tiles (e.g., [24, 38]), in Dragonfly, tiles may be skipped, or fetched out-of-order.

Dragonfly is different from recent work that transmits streams in two qualities [26, 43]: (i) its utility driven scheduling allows for the primary tiles to be skipped or transmitted at different qualities, unlike [43] which transmits the same quality for all primary tiles combined with passive skipping (we refer to this scheme as Two-tier); and (ii) it does not require real-time encoding unlike [26].

Although transmitting a masking stream incurs overhead, this is small because (i) the masking stream has lower quality; and (ii) transmitting the masking stream allows Dragonfly to reduce the overall rate needed for the primary stream. This is because having the masking stream reduces the penalty incurred when the viewport moves and a portion of the primary stream is not available. Thus, it is possible to predict what to send closer to the playback deadline (which is more accurate) and reduce the size of the window to be fetched in the primary stream around the predicted viewport. Thus, the increase in masking overhead is compensated by a decrease in overall primary rate.

Validation with user study and emulations. We have developed a prototype implementation of Dragonfly[1], and three state-of-the-art approaches: Flare [38], Pano [24], and Two-tier [43]. We integrated all these systems with an Oculus headset, which enables users to experience 360° videos under different algorithms. Our key results are:

- An IRB approved user study with 26 users shows that when streaming a variety of real 360° videos, users report a rating of 4 or higher (Good/Excellent) for 65% of the sessions, while only 16% of sessions with Pano, and 13% of sessions with Flare achieve this rating.
- Experiments on an emulation testbed with real bandwidth traces show that Dragonfly achieved median PSNR gains of 1.72dB, 2.5dB and 4.5dB over Flare, Pano and Two-tier, respectively despite proactively skipping primary tiles. Further, 99% (resp. 50%) of Flare (resp. Pano) sessions experienced rebuffering but no Dragonfly session experienced incomplete frames owing to its use of a masking stream.
- An ablation study shows that Dragonfly significantly outperforms a variant that uses a masking stream combined with passive skipping, achieving median PSNR gains of 1.6 dB. Dragonfly skips more tiles in primary stream (6.74% for Dragonfly vs 2.17% for the variant), but performs better because it fetches more tiles in the highest quality (83.4% for Dragonfly vs 53.6% for the variant). Finally, a variant without masking achieves median PSNR comparable to Dragonfly thanks to its proactive skipping, but may suffer from incomplete frames.

2 BACKGROUND AND MOTIVATION

Background: tile-based streaming. Many traditional video publishers (e.g. YouTube, Facebook, and Vimeo) treat 360° video like conventional video streaming. They split the video into chunks of short duration (e.g., 1 second) and encode each chunk at multiple qualities, with each quality capturing the entire 360° view. They then use traditional Adaptive Bit Rate (ABR) algorithms [18, 27, 28, 33, 41, 44, 49] to stream the video to the client. These algorithms decide the level of quality at which to fetch each chunk so as to ensure high overall quality, while minimizing client rebuffering. This approach ensures responsiveness to user motion, but results in

[1]See https://github.com/Purdue-ISL/Dragonfly for artifacts including source code.

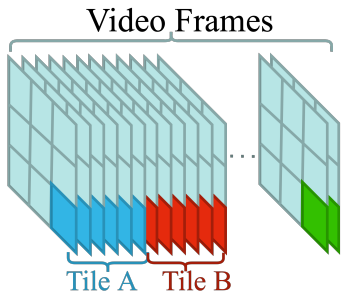


Figure 1: Tile-based streaming: Video is split temporally into chunks, which are partitioned spatially into tiles.

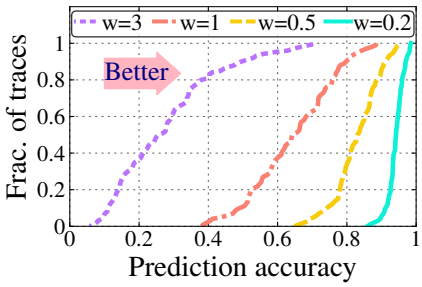


Figure 2: Accuracy of viewport prediction (i.e, fraction of tiles in viewport that are predicted) degrades sharply with larger prediction windows.

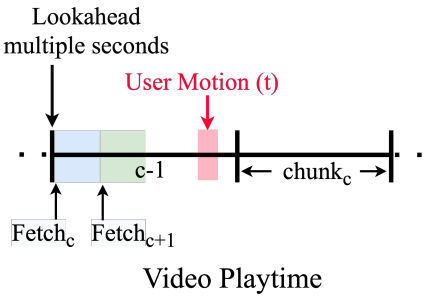


Figure 3: Looking ahead multiple chunks results in sub-optimal quality enhancement.

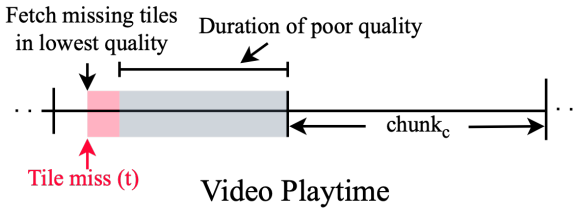


Figure 4: Fetching tiles close to a deadline results in poor quality that persists.

significant wasted bandwidth, and degraded quality in bandwidth constrained environments.

Much recent research has sought to tackle these challenges [19, 21, 24, 31, 37–39, 42, 51]. Rather than the full 360° view, these approaches seek to stream the views of relevance to the user. Many of these approaches rely on *tile-based* streaming (Figure 1). In this approach, each video chunk is partitioned into *tiles*, where a tile has the same number of frames as the original chunk but covers only a smaller spatial region of the frames. Tiles may be fixed size (e.g., forming a 4×6 grid [38]) or variable sized (e.g., [24]). Each tile is encoded at different qualities, and tiles can be independently downloaded and encoded. Tile-based approaches (i) *predict the viewport* of the user in the near future based on a history of the recent viewports; and (ii) fetch tiles based on their relevance to the user. These *view-centric* approaches significantly reduce bandwidth requirements for streaming, and can deliver higher quality in bandwidth-constrained environments.

Challenges for view-centric approaches. View-centric approaches must confront two main challenges:

Simultaneously dealing with motion and network uncertainty. View-centric approaches must decide how far ahead of playback time they should predict the viewport (the *prediction window*). Figure 2 shows that prediction accuracy degrades sharply with prediction window – median accuracy is 94.2% for a window of 0.2 seconds and only 25.4% for a window of 3 seconds. We obtained these results using linear regression (shown to perform well in [24, 38]) on user traces from [34].

Prediction inaccuracy can trigger *motion-induced stalls*. To minimize such stalls, these approaches must use small prediction windows (or, equivalently, maintain small client buffers). However, smaller buffers can make clients vulnerable to dips in network bandwidth, resulting *network-induced stalls*.

To avoid this, some systems [21, 24, 37] fetch the full 360° for every chunk, but fetch tiles for each chunk at different qualities based on their predicted importance. Since they use a look-ahead of multiple chunks, and never revisit their decisions, deciding tile qualities too early can result in enhancing the quality of the wrong set of tiles. This is illustrated in Figure 3, where a client fetches chunks c and $c+1$ while playing back $c-1$. Unfortunately, user movement after the client makes these decisions may change the relative importance of different tiles corresponding to chunks c and $c+1$.

Other systems [19, 31, 38, 39, 51] fetch tiles in the predicted viewport, and additional tiles around it. Erroneous predictions can result in motion-induced stalls. Flare [38], a notable system in this class, continually refines predictions and may later fetch additional tiles discovered to be necessary for playback. Unfortunately, it must fetch the missing tiles with a more stringent deadline, so that these are fetched at lower rate leading to lower quality that persists for several frames. To illustrate, consider Figure 4. Here, Flare misses a tile at time t close to the start of a chunk boundary. To meet the deadline requirements, it fetches a low quality version of the tile. Unfortunately, this lower quality version persists for the entire chunk duration.

Stalling playback hurts interactive experience. Existing schemes treat 360° streaming similar to traditional Video on Demand (VoD): they pause the playback of video until *all tiles* in the user's current viewport arrive, resulting in stalls (rebuffering events). However, this approach can impair user interactive experience. For instance, a user may move during the playback pause, leading to a situation where the newly arriving content is stale. This can result in further stalling video playback and requiring fetches of new content, prolonging rebuffering time. A snapshot of a real user trace of yaw movement (from the user study described in §4.5) in Figure 5 shows that users can move significantly while a video is rebuffering.

3 DRAGONFLY DESIGN

Dragonfly is designed to address the limitations of existing view-centric approaches. Dragonfly leverages two observations: (a) stalling

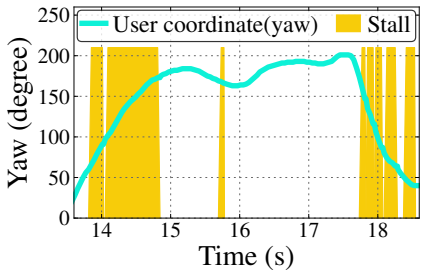


Figure 5: Users can significantly move (changes in yaw) during stall events (blue regions).

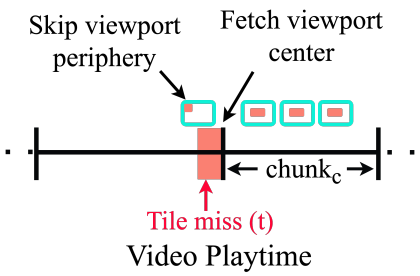


Figure 6: Skip a tile in viewport periphery, and needed for few frames. Instead fetch tiles in center and needed for more frames.

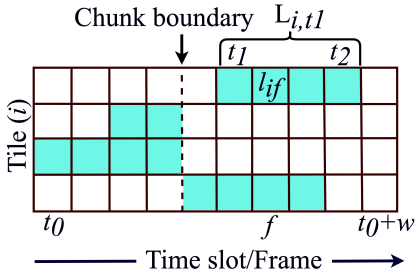


Figure 7: Computing location score in the look-ahead window.

playback waiting for *all tiles* in the current viewport hurts interactive experience; and (b) continuous playback can be enabled by transmitting a low quality masking stream. However, user experience critically depends on fetching an appropriate set of viewport tiles in the primary stream prior to the rendering deadline.

In deciding how to schedule tile fetches in the primary stream, existing approaches [24, 38] cannot be directly applied, as they are designed to fetch all tiles while minimizing stalls. A potential solution is to employ these algorithms and simply skip tiles that do not arrive by the deadline (a *Passive Skip* strategy) so as to achieve continuous playback.

Instead, Dragonfly is motivated by the observation that the freedom to *proactively* skip tiles enables careful *prioritization* of tile downloads, taking into consideration factors other than tile deadlines. More specifically:

- Dragonfly can skip tiles at the viewport periphery to instead more quickly retrieve tiles at the center that may be needed in the near future (Figure 6). In contrast, fetching less critical tiles may delay more critical tiles, leading to further skips or degraded quality for these tiles.
- Dragonfly prioritizes tiles that are expected to benefit more frames over tiles only needed for a few frames. For instance, in Figure 6, it would skip a tile close to the chunk boundary, and instead prioritize a tile at the start of the next chunk, which would be needed for more frames.
- Dragonfly can choose to download a tile at a higher quality *even* if it may add delay. For instance, consider Figure 4 again. It may improve net perceptual quality to fetch a higher quality version of the tile if doing so can benefit sufficient frames, even if the tile has to skipped for the first few frames. In contrast, systems that stall playback fetch low quality versions to minimize rebuffering (§2).

Like other systems [24], Dragonfly may also prioritize tiles whose marginal quality difference between higher and lower qualities is higher (Figure 18 in Appendix §A.).

Since Dragonfly uses two streams, and can proactively skip frames in the primary stream, it has the flexibility to defer decisions on which primary tiles to fetch closer to the playback deadline. This is advantageous since predictions of user motion are more accurate closer to playback deadline (§2). In contrast, systems that use a single stream do not have this flexibility and may need to make fetch decisions further from playback to protect against network uncertainty.

Dragonfly's decisions on which tiles to fetch are constrained by the available network bandwidth (obtained from the throughput predictor [49]). To obtain the highest possible quality subject to this constraint, Dragonfly uses a utility formulation (§3.1) and allocates bandwidth across the primary and masking streams (§3.2).

3.1 Utility-Driven Scheduling algorithm

In this section, we describe the Dragonfly scheduling algorithm, which decides which tiles to fetch, in what quality, and in what order. To motivate our approach, we first discuss the factors that affect the relative importance of tiles to be fetched.

The first factor is *location score* (l_{if}), which captures the predicted importance of displaying tile i when the user views frame f. Consider Figure 7 which illustrates when the regions associated with four different tiles are in the user's viewport. In this example, the topmost tile is relevant to the user between time t_1 and time t_2. Even though the tile may lie within the user's viewport during the period (t_1, t_2), it may be of greater importance during certain time intervals (e.g., when the associated region is at the center of the user's viewport during that interval). The second factor, *cumulative location score* (L_{it}), captures the total location score when a tile is fetched by time t. Consider Figure 7 again, the topmost tile offers maximum value if fetched prior to t_1, and the value progressively decreases until time t_2 beyond which fetching the tile offers no value. The third factor, *sensitivity to quality* (Q_{iq}), captures the quality metric score for tile i at encoded quality q. A higher quality for the tile produces greater value to the user, but the relative benefit of fetching a higher quality depends on video content itself.

Utility-based Approach: Overview. Dragonfly captures these factors through a function that assesses tile utility based on when the tile is fetched, how central it is to the user's experience at different points in time, and the tile quality. It uses these utility values to determine which tiles to fetch and in what sequence and quality.

More formally, consider a discrete time model where each unit of time represents the playback duration of one frame. Time t denotes that the playback of $t-1$ frames has been completed, and the playback of the t^{th} frame is currently in progress. At time t_0, when computing utilities, Dragonfly considers predicted viewports during

the window of time $(t_0, t_0 + W)$, where W represents a look-ahead period. For each tile i, Dragonfly computes:

Location score. The location score assesses the importance of tile based on its location in the viewport. To tolerate errors in the prediction of the future viewport, and to capture that certain spatial regions (e.g., the center) may be more important even within the predicted viewport, Dragonfly predicts multiple concentric regions of interest (RoI). The innermost RoI is close to the center of the predicted viewport, while the outermost RoI includes both the predicted viewport, and a region just outside the viewport. The location score is $l_{if} = \sum_{r=1}^{C} l_{irf}$, where l_{irf} is the degree of overlap of the region corresponding to tile i, and the RoI r. We choose $l_{irf} = 1$ if the spatial region of tile i is completely within the RoI r, $l_{irf} = 0$ if there is no overlap and otherwise l_{irf} is set to a fractional value.

Cumulative location score. Next, we compute L_{it}, the cumulative location score if tile i is fetched by time t. Precomputing the cumulative scores for different tile arrival times reduces the computations to be performed by the scheduling algorithm, as we will see later. For computations in the time window $(t_0, t_0 + W)$, the cumulative location score is $L_{it} = \sum_{f=t}^{t_0+W} l_{if}$. Figure 7 illustrates the computation.

Sensitivity to quality. When deciding what quality to use for each tile, Dragonfly accounts for the fact that higher quality encoding may be more critical for some tiles than others. In Figure 18 in Appendix §A, we show an example of how one tile may exhibit more sensitivity to quality than another tile. Q_{iq} can set based on any quality metric (e.g., PSNR, SSIM, or PSPNR [24]).

Tile utility. Dragonfly uses both the cumulative location score, and quality difference to calculate tile utility. Specifically, it computes U_{iqt}, the utility of receiving tile i by time t with quality q as:

$$U_{iqt} = f(L_{it}, Q_{iq}) = L_{it} Q_{iq}$$

The current implementation simply takes the product of the two scores; future work can explore other choices of f.

Formalizing Tile Scheduling. We now describe an optimization formulation to capture the scheduling problem that Dragonfly solves at time t_0, and over a look-ahead window W. This optimization decides which tiles to fetch, in what qualities, and when, given a bandwidth estimate B over the look-ahead window, while optimizing the overall utilities.

$$\max_{z} \sum_{i=1}^{C} \sum_{t=t_0+1}^{t_0+W} \sum_{q=0}^{Q} U_{iqt} \cdot z_{iqt}$$

$$\text{s.t.} \sum_{i=1}^{C} \sum_{q=0}^{Q} \sum_{j=t_0}^{t} S_{iq} \cdot z_{iqj} \le B(t - t_0) \quad \forall t_0 \le t \le t_0 + W$$

$$\sum_{t=t_0}^{t_0+W} \sum_{q=0}^{Q} z_{iqt} = 1 \ \forall i$$

$$z_{iqt} \in \{0, 1\} \ \forall i \ \forall q \ t_0 \le t \le t_0 + W$$

Here, S_{iq} is the size of tile i if fetched in quality q, and U_{iqt} is precomputed for all tiles, C, needed for the predicted viewports in the window. The binary variable z_{iqt} is 1 if tile i is fetched at quality q, and arrives at time t, and 0 otherwise.

The first two constraints in the formulation respectively ensure that:

- the total data that arrives by each time step t in the look-ahead window cannot exceed bandwidth constraints;
- each tile is associated with exactly one quality and can arrive at exactly one time.

Quality $q = 0$ indicates the tile is skipped (and the corresponding S_{iq} is 0). Utility may be non-zero even if the tile is skipped: when the algorithm is used for the primary stream, skipping a tile in this stream implies displaying the masking version of the tile.

A Greedy Heuristic. The formulation above is an Integer Program, with the number of decision variables depending on the number of tiles that must be scheduled, the number of quality choices, and the length of the look-ahead window. For tractability, we use a greedy heuristic.

Our algorithm starts by initializing the utility of each tile to the utility associated with the low quality masking version of the tile (if it arrived by t_0), or 0 otherwise. It maintains a list (fetch list, initially empty) of all tiles that should be fetched by the primary stream, and the associated quality. The algorithm proceeds in a series of rounds, where in each round a subset of tiles may be promoted to a higher quality. The process is not monotonic — it is possible that some tiles have their quality reduced as we will discuss.

In each round, the algorithm first computes the utility gain of promoting each tile from its current assigned quality to quality q. It considers tiles in the order of highest utility gain. For each considered tile i, the algorithm considers the impact on total utility if it upgrades the tile's quality to q, and inserts it in each possible position of the fetch list. This involves for each prospective list position, computing the arrival time of the tile based on a bandwidth estimate, adjusting the arrival time of subsequent tiles in the fetch list, and recomputing utilities for all these tiles.

The algorithm either inserts the tile with quality q in the best position that maximizes total utility, or sticks to the current quality if no position improves the total utility. Inserting a tile in a list may imply that subsequent tiles miss their deadlines and have utilities adjusted to zero, which is acceptable as the insertion increases the total utility. In such cases, the algorithm demotes the quality of the tile iteratively until is completely dropped from the fetch list if it continues to have a zero utility for the lowest quality in the primary stream. The scheduling algorithm has a complexity of $O(C^2Q)$, where C is the number of tiles relevant to the look-ahead window, which we find acceptable in practice. The pseudo-code is summarized in Algorithm 1 in the Appendix.

3.2 Masking stream transmission

To mask skipped tiles in the primary stream, Dragonfly transmits a low quality version of the tiles as part of the masking stream. It sends this stream with a longer look-ahead, since this stream helps decouple network from motion uncertainty. The masking stream helps tackle missed tiles owing to dips in bandwidth, while the primary stream more accurately fetches content relevant to the user as it uses a small look-ahead.

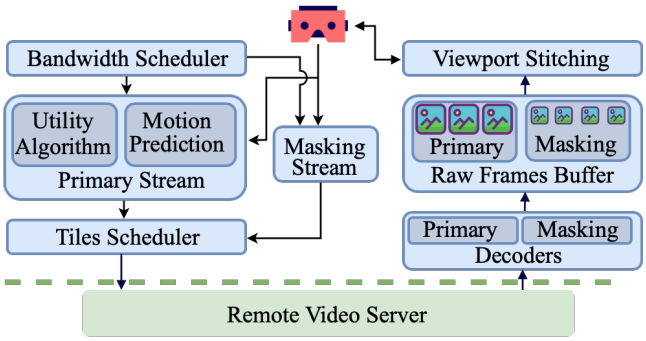


Figure 8: Dragonfly overview.

A larger look-head associated with the masking stream implies higher motion uncertainty, so it must include a wider region around the viewport. We have implemented two strategies for the masking stream. A first implementation involves simply transmitting the full 360° for the masking stream (without tiling). A second implementation uses tiles for the masking stream, where the maximum displacement around the user's coordinate that the masking stream must fetch is a configurable parameter that can be specified on a per chunk basis.

Our experiments show the two schemes perform comparably, although the full 360° approach performs slightly better (Figure 19 in Appendix A). In particular the tiling approach sees slightly more incomplete frames, and interestingly has slightly more overhead. This is because although a smaller region is transmitted, the encoding overheads of a tiling approach especially at low qualities is higher.

For this reason, our experiments focus on sending the full 360° for the masking stream. That said, we note that multiple optimizations may be implemented for the tiling approach in the future. A first optimization is to use the scheduling algorithm in §3.1 to ensure the decision of which masking tiles to skip is done carefully based on the utility function. A second optimization is to mask the effects of a missing tile in the masking stream by interpolating neighboring tiles.

3.3 Implementation

Figure 8 presents an overview of Dragonfly. We implemented a prototype of Dragonfly in C++ with 7500 lines of code on a Linux platform. The key components are (i) a tile scheduler which decides the order in which tiles should be fetched, and which tiles may be dropped even if present in the viewport; (ii) a bandwidth scheduler that decides how to split traffic between the primary and masking streams. Other components include (i) decoders which decode both primary and masking tiles, (ii) viewport constructor which stitches tiles together for rendering, and (iii) predictors for user motion and bandwidth.

The client uses linear regression to predict viewports tiles like prior work [24, 38]. It sends a request with a list of tiles along with the desired quality per tile to the server. When a tile is received, the client calculates the receive time for future bandwidth estimation, and decodes the tile using the `ffmpeg-libavcodec` C++ library [5]. For decoding, we allocate an in-memory decoder buffer using `avio_alloc_context` [6]. For rendering, the client stitches all viewport tiles and replaces missing tiles with black pixels. The server is a modified version of a DASH server. The manifest file is updated to include tile sizes, the quality metric for that tile (e.g., PSNR or PSPNR) for all quality levels, and the yaw and pitch displacements on a per-chunk basis.

Since the client can send multiple requests for the same tile, the server tracks the quality per tile sent, and only sends a tile redundantly if it was previously fetched with masking quality. The client refreshes its list of tiles over time as it periodically refines its predictions. When a new request is received, the server discards the previous (older) request, and transmits tiles as per the newer request. Tiles not yet transmitted in the older request in the send queue are dropped.

4 EVALUATION

We use both emulation (§4.3) and user studies (§4.5) to compare Dragonfly with state-of-the-art 360° streaming approaches. We also dive deeper into Dragonfly design choices by conducting an ablation study (§4.4).

4.1 Metrics and schemes compared

We evaluate the performance of Dragonfly and other 360° systems using both subjective and objective metrics. Our user studies report on the **Mean Opinion Score (MOS)** provided by the users. Our objective metrics include (i) *Quality metric*, primarily **PSNR**, with **PSPNR** used in some experiments; (ii) **Rebuffering ratio,** i.e., the ratio of the total time the session experiences rebuffering to the total video playtime; (iii) **Incomplete frames %**, the percentage of viewports per session with at least one skipped tile[2]; (iv) **Blank area**, the fraction of viewport area that is blank; (v) **Bandwidth wastage,** defined as the ratio of the unnecessary data received by a system to the total data that it receives, where unnecessary data corresponds to tiles that are either outside the actual viewport or are within the viewport but are not rendered. (e.g., a redundant masking version is unnecessary data). Since the masking stream may contain a full 360° chunk rather than a tile, if only a portion of it is relevant, we consider the useful data to be the minimum of the bytes needed to encode the relevant area using a tiling approach, or the bytes to encode the full 360°, whichever is smaller.

We compare Dragonfly with several schemes:

Flare. Flare [38] fetches the viewport tiles, and additional tiles at the periphery (§2). It continually refines predictions, and may fetch additional tiles with a quality that depends on the bandwidth and playback delay constraint.

Pano and Pano-PSPNR. Pano [24] uses a traditional ABR algorithm to first determine a bitrate for each chunk. Next, given the bitrate, Pano assigns a quality level for each tile within the chunk so as to maximize a desired quality metric. Further, all tiles in the 360° are sent with non-viewport tiles assigned the lowest quality. By default, our experiments, and user study focus on a Pano variant which maximizes PSNR. We have also conducted evaluations with a version of Pano that optimizes PSPNR, a metric that adjusts the

[2]Most existing 360° approaches suffer from rebuffering when tiles that are needed are not received. Dragonfly suffers from skipped tiles instead.

Scheme	# of streams	Refine fetch decision	Skip/stall approach
Dragonfly	Two	Yes (100ms)	Utility
Two-tier	Two	No	Stall/Passive
Pano	One	No	Stall
Flare	One	Yes (100ms)	Stall

Table 1: Comparing schemes by their design choices.

PSNR calculation accounting for the fact users are less sensitive to quality distortion for certain kinds of video content (§4.3).

Pano introduces and uses a new quality metric (PSPNR-360) that extends traditional PSPNR to also consider factors such as user viewport speed, luminance and Depth of Field. We do not consider this new metric for a few reasons. First, the choice of quality metric is orthogonal to our comparisons since Dragonfly may also use the same metric by incorporating it into the utility function. Second, calculating PSPNR-360 is non-trivial, and requires offline calculation of the metric for a variety of user movement patterns, and an online estimation of patterns for a given user. Further, the code for PSPNR-360 is unavailable to us[3].

Two-tier [43]. Like Dragonfly, this system has a low quality 360° stream, and a higher quality primary stream. Unlike Dragonfly's proactive skip approach, Two-tier fetches all primary stream tiles in the same quality and passively skips those that do not arrive in time. Further, Two-tier stalls if not all viewport tiles of the masking stream arrive in time.

Table 1 compares systems with respect to their design choices. Note that Two-tier and Pano decide what quality to fetch tiles once per chunk, and do not refine these decisions. In contrast, Dragonfly and Flare make an initial fetch decision, but refine it every 100ms. This may result in previously requested tiles being canceled if not already transmitted, and new tiles requested instead.

Scheme implementations: Since the source code is not publicly available for Flare and Two-tier, and for key parts of Pano as discussed above, we implemented them by modifying the Dragonfly codebase. (§3.3). For Pano and Flare which use a single stream, we used a look-ahead of 3 seconds (but perform sensitivity to this choice). For Two-tier and Dragonfly, we used a look-ahead of 3 seconds (resp. 1 second) for the masking (resp. primary) stream.

4.2 Datasets

360° videos dataset. Our experiments are conducted using a subset of the videos from a publicly available dataset [34]. The dataset has 30 videos (each 1 minute long and associated with a set of user movement traces) which differed in terms of the levels of camera motion, and number of moving objects. We spatially divided the videos into 12x12 tiles (see Appendix §A for a discussion of tiling schemes).

We encode the videos with Quantization Parameters (QP) of 22 (highest quality), 27, 32, 37,and 42. The bitrates vary with both the videos and qualities and across chunks. For QP 42, the median bitrate of videos varied from 0.9 Mbps to 4.6 Mbps. For QP22, the median video bitrates varied from 10.4 Mbps to 49.6 Mbps (and the median of these medians was 28 Mbps). Table 3 and Figure 24 in the Appendix provide detailed information for all videos and quality levels. For Two-tier and Dragonfly which use two streams, the lowest quality is used for one stream, and the remaining 4 qualities were used for the other stream. All other systems were given the 5 qualities for their single stream.

We used `ffmpeg-crop` to segment the video into tiles. Then, we encode the video tiles with different QPs to generate multiple qualities of the same video tile. We set the I-interval (i.e. chunk length) to 1 second, and used ffmpeg to divide the video into chunks. To generate the PSNR values per tile, we used video quality management tool (VQMT) [14] and Pano's scripts to generate PSPNR [11].

Network traces. We use throughput traces from two datasets: (i) **Belgian dataset** [45], which has 40 different bandwidth logs with a total length of 5 hours collected by downloading large files using HTTP over a 4G network when the mobile user was using different means of transportation (train, bus, car, or foot); and (ii) **Irish dataset**: [40], from which we selected 5 bandwidth logs which were collected over 4.3 hours by repeatedly downloading a large file (>200MB) over a 5G network while users were static.

For both datasets, we crop logs into 1 minute traces. We filter out traces if (i) their bandwidth was so high that the full 360° could be streamed in high quality for significant portions of the session or (ii) if their bandwidth was insufficient to stream just the viewport in the highest quality most of the time. Specifically, we filtered out traces if the $10^{th}\%ile$ bandwidth is less than 7 Mbps and the $90^{th}\%ile$ bandwidth is greater than 28 Mbps[4]. These numbers were chosen as the median of the median video bitrates is 28 Mbps as noted above, and streaming only the viewport requires about 25% of the full 360° bandwidth. Individual bandwidth samples in some filtered traces (especially in the Irish dataset) could still be high, so we cap each bandwidth sample to 28 Mbps. We plan to validate with "in-the-wild" experiments over the Internet in the future [47, 48].

4.3 Comparison of schemes

Figure 9 shows results comparing the systems listed in Table 1 with 770 end-to-end video streaming sessions using all combinations of 7 videos, 10 user traces, and 11 bandwidth traces from the Belgian dataset. Comparisons are given in terms of PSNR and rebuffering ratio/incomplete frame %.

From Figure 9(a) we make several observations. First, Dragonfly performs the best in PSNR. The median PSNR is higher relative to Flare and Pano by 1.72 dB and 2.5 dB respectively. Note that even a 1dB PSNR increase is considered significant in the video coding community (and corresponds to 20% lower mean square error), while a 3dB increase corresponds to 50% less mean square error. Second, although Two-tier rarely sees rebuffering (expected since it fetches the full 360° chunk in lowest quality three seconds in advance), the PSNR is the lowest. This is because the scheme picks a uniform quality for all tiles in the enhancement stream, unlike Dragonfly which uses a utility algorithm to fetch the most important tiles with higher quality. Third, sessions with Dragonfly

[3] [11] provides scripts for grouping tiles using precomputed PSNR-360 models for videos used by Pano. However, code is unavailable for key parts including calculation of PSPNR-360 (for the offline lookup table, and for each user online), and for the assignment of quality to tiles for a given chunk bitrate. We reused the grouping script, but group tiles by PSNR or PSPNR. We implemented quality assignment using the paper's ideas.

[4] For the Irish dataset, this filtered out too many traces. Instead, we filtered if $75^{th}\%ile$ exceeded 28 Mbps

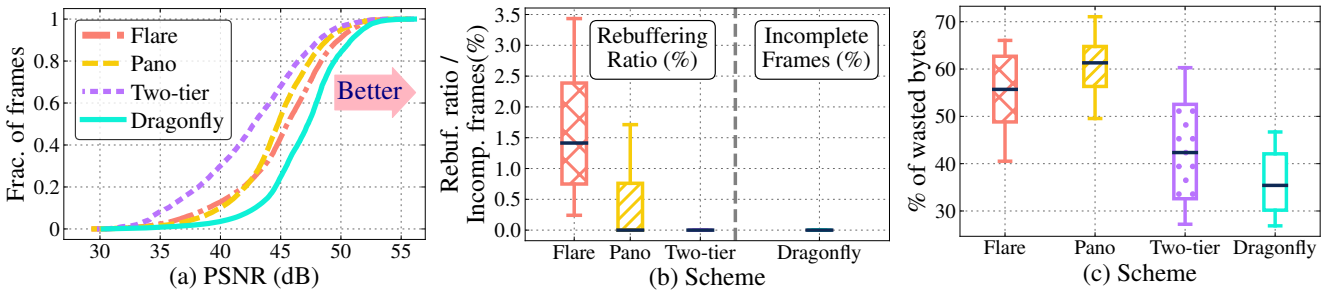


Figure 9: Objective quality metrics. (a) Distribution of PSNR across viewports of all sessions; (b) Rebuf. ratio/ Incomp. frame % across sessions; (c) Percentage of wasted bytes.

do not see missed tiles since it fetches the full 360° for masking. In contrast the median rebuffering ratio across sessions with Flare is 1.42%. While Pano has a lower median rebuffering, the spread is large, with 90%tile rebuffering ratio of 1.71%.

Bandwidth wastage. Figure 9(c) shows the bandwidth overhead (percentage of wasted bytes) for each system. Surprisingly, Pano and Flare see higher overheads (61.31% and 55.7% for the median session, respectively), while Dragonfly and Two-tier see lower overheads despite fetching a redundant masking stream. This is because schemes with a single stream (Pano and Flare) fetch tiles with a larger look-ahead of 3 seconds to provision for network dips. This results in prediction errors that can affect quality (e.g., viewport tiles may be fetched in lower quality while tiles outside the viewport are in higher quality).

We also evaluated alternate versions of Pano and Flare with a 1 second look-ahead. For Flare, the overheads do drop noticeably to 38.31% for the median session. However, for Pano, the overheads only drop to 57.45% for the median session. This is because Pano groups tiles with similar sensitivity to quality variations (our implementation groups based on PSNR or PSPNR rather than PSPNR-360). All tiles in a group are fetched with the same quality. Pano does so because of the potential benefits of more effective compression with the larger grouped tiles. However, this benefit is outweighed by the cost of sending a larger region outside the viewport at a higher quality. Further, we find that the compression efficiency benefits of larger tile sizes while notable for low qualities degrade significantly at higher qualities (we discuss this further in the Appendix §A). Finally, Two-tier has a higher overhead than Dragonfly because it does not refine its fetch decision which leads to higher inaccuracies, and hence higher overhead.

Alternate quality metric: PSPNR. While Dragonfly optimizes for PSNR, it can also work with other quality metrics (§3.1). To demonstrate, we compare variants of Dragonfly and Pano that optimize for PSPNR using the same set of videos, user trajectories, and bandwidth traces described above. Figure 10 shows that Dragonfly-PSPNR performs better than Pano-PSPNR. Further analysis shows Dragonfly achieves a higher PSPNR across all viewports, and improves PSPNR by over 2 dB for 69% of viewports. We do not show

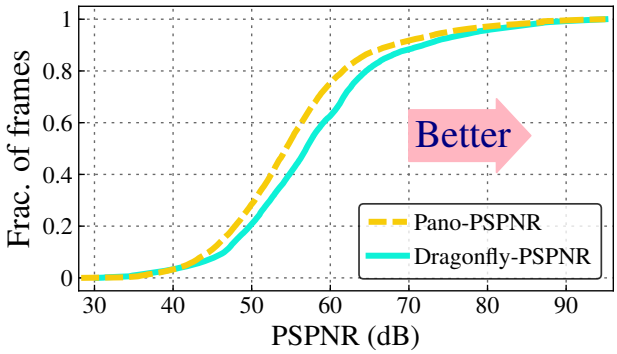


Figure 10: Variants of Pano and Dragonfly optimizing for PSPNR.

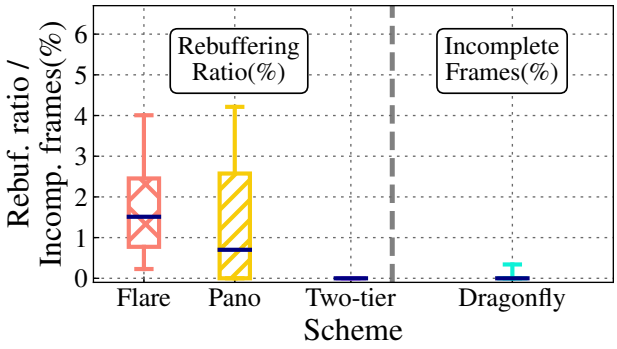


Figure 11: Sensitivity analysis using bandwidth traces from Irish dataset.

the performance with respect to rebuffering ratio and incomplete frames as it is similar to our earlier comparisons.

Sensitivity to bandwidth traces. Our experiments so far have been conducted with the Belgian traces. We next present results with the Irish dataset. We used 10 traces, along with the same set of 10 user trajectories and 7 videos earlier. We conduct experiments for all combinations and summarize results in Figure 11. Overall, the general trends are similar to the Belgian traces, although the performance is slightly worse in all metrics for all schemes. Interestingly, Pano sees higher rebuffering ratios with these traces. Further investigation shows that bandwidth in these traces exhibits abrupt occasional dips where bandwidth is close to zero. While this affects Pano and Flare, Dragonfly is more resilient to the dips since it uses a masking stream.

Sensitivity to user movement. In Appendix A, we present results exploring the sensitivity of Dragonfly to motion prediction errors. Results show that Dragonfly achieves higher PSNR relative to other schemes under various ranges of error, while maintaining the lowest overhead.

Scheme	# of streams	Refine fetch decision	Skip/stall approach
PassiveSkip	two	100ms	Passive
PerChunk	two	chunk	Utility
NoMask	one	100ms	Utility

Table 2: A summary of Dragonfly variants.

4.4 Dragonfly ablation study

The main benefits of Dragonfly arise from three design elements: (i) use of two streams; (ii) a utility-driven proactive skip approach; and (iii) refining fetch decisions every 100 ms, rather than making them once per chunk. We next present an ablation study where we compare Dragonfly with three variants: PassiveSkip, PerChunk and NoMask (summarized in Table 2) that each drop one of the elements to understand the impact on performance. We use the same set of videos, and user traces and use the Belgian bandwidth traces.

Figure 12 summarizes the performance of Dragonfly variants. Figure 12(a) shows that Dragonfly streams the highest viewport quality as expected. The median PSNR of Dragonfly is 4.8dB and 1.6dB higher than PerChunk and PassiveSkip, respectively. PerChunk has the lowest PSNR among the variants indicating the importance of refining fetch decisions. Next, PassiveSkip achieves significantly less PSNR than Dragonfly. This shows that masking alone is insufficient, and the proactive skipping strategy that Dragonfly adopts to decide which primary tiles to fetch is essential. Interestingly, NoMask performs comparably to Dragonfly on the median PSNR. However, it exhibits a significant tail shown in the zoomed portion of Figure 12(a). This is because NoMask has a small % of incomplete viewports which impacts the PSNR tail. Note that for skipped masking tiles, we calculate and use the PSNR of black tile.

Figure 12(b) shows the fraction of blank area per viewport across schemes. NoMask is the only variant with incomplete viewports, and about 10% of the viewports are incomplete with this scheme. This is expected because NoMask is the only scheme that does not transmit a masking stream. Note that even when a viewport is incomplete, Dragonfly's proactive skip algorithm ensures more critical regions of the viewport are fetched, which limits the impact on PSNR for the vast majority of frames. Finally, Figure 12(c) shows that NoMask sees the lowest overheads in terms of wasted bytes. This is expected because it does not fetch the masking stream.

Proactive vs Passive Skip. We further analyze the benefits of Dragonfly's proactive skip algorithm. Figure 13(a) shows the % of skipped tiles in the primary stream with various schemes. Interestingly, Dragonfly skips noticeably more tiles than PassiveSkip when the primary stream is considered – 39% of Dragonfly viewports have skipped primary tiles in comparison to 7% of PassiveSkip viewports. Yet, as Figure 12(a) has shown, Dragonfly performs better in PSNR. This is because Dragonfly fetches more critical tiles in higher quality (e.g., central tiles that benefit more frames), while proactively skipping less critical tiles (e.g., tiles in the periphery and which may only benefit a small number of frames). Figure 13(b) explores this further by showing the fraction of viewport tiles received in each quality. The figure confirms that (i) Dragonfly fetches more tiles in masking quality (6.74%) relative to PassiveSkip (2.17%); but (ii) Dragonfly fetches significantly more tiles in the highest quality (83.4%) compared to PassiveSkip (53.6%). This indicates the importance of Dragonfly's proactive skip algorithm. Finally, PerChunk skips a noticeably more primary tiles (45.72%) because it does not refine its fetch decision.

4.5 User Study

In this section, we report on an Institutional Review Board approved user study that we used to compare Dragonfly with other systems.

Ethical considerations and study protocol. Our user study was conducted using a protocol approved by Purdue University's IRB. Each participant was asked to view a series of five 360° videos using an Oculus headset. Video was streamed emulating different bandwidth conditions using Mahimahi [35]. Five bandwidth traces were selected from the Belgian dataset, and for any given participant and video, one of the traces was randomly assigned. Each video was streamed with the selected bandwidth trace using each of three approaches: Pano, Flare, and Dragonfly. The order of the videos was randomized for each participant, and the order in which the three systems were used to stream each video was also randomized. This avoided systematic biases that may arise owing to a user always viewing a video for the first time with a particular system. In total, each participant viewed and evaluated 15 videos of 1 minute length with estimated experiment duration of 45-60 minutes. The participants rated their experience with each video streamed with each approach on a numeric scale from 1 (bad) to 5 (excellent) [13], and optionally provided subjective comments.

Participants were encouraged to move freely so the video streamed reflected their own motion. This is in contrast to many prior studies. For instance, Flare [38] obtains user head movement traces using a commercial 360° system, but all comparisons use emulations. Pano [24] compares schemes using user trajectories on an emulation testbed, and the actual user study involves participants passively watching videos obtained from emulation runs with the different schemes. We conducted the study while allowing participant movement to ensure that the evaluation better reflects end-to-end user experience.

Prior to viewing the first video for which we record the rating, users were instructed on how to use the HMD, and requested to view a trial video recorded with poor and good quality, so they could get familiar with the HMD device and be properly calibrated on the range of qualities they may experience. We recorded how the participant interacts with the videos by logging the participant's head movements when using the HMD device. All data regarding participants were recorded with an anonymous participant ID. Although the risk of motion sickness is minimal, participants were advised through an informed consent form to not participate if they had a history of motion sickness. Further, they were encouraged to

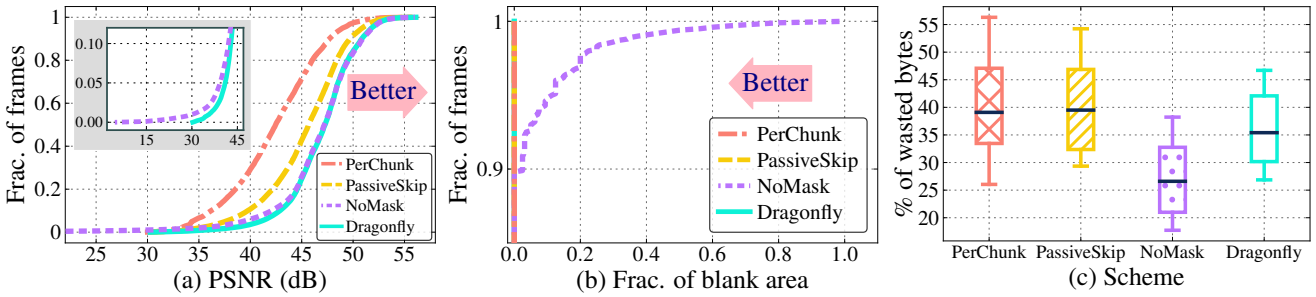


Figure 12: Results of Dragonfly variants. (a) Distribution of PSNR across viewports of all sessions; (b) Fraction of blank area per viewport; (c) Percentage of wasted bytes.

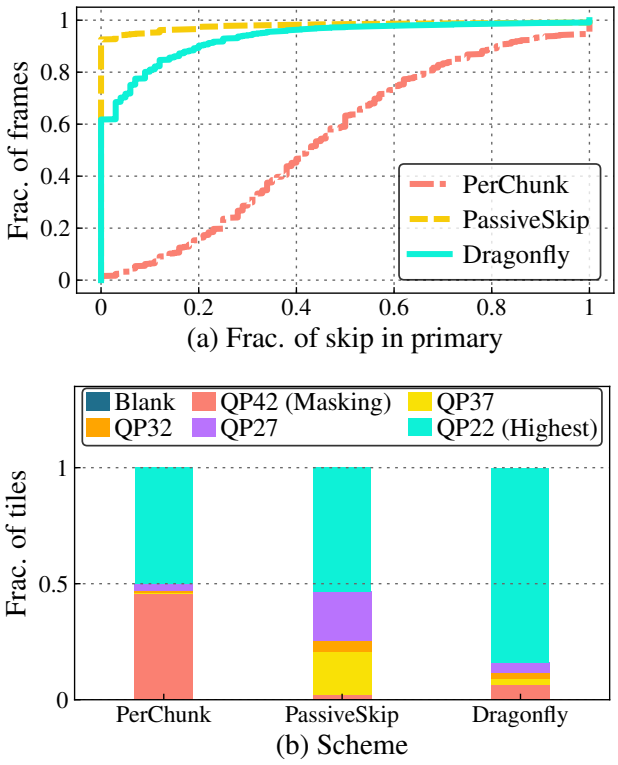


Figure 13: The fraction of tiles skipped in primary stream, and its impact on the quality of viewport tiles.

take periodic breaks, and not required to watch all videos. Additionally, we first viewed the videos in our dataset prior to the study, and conservatively avoided two videos used in our emulation experiments that we felt had the potential to cause mild motion sickness including our highest bitrate video. Conducting user studies with high motion videos ethically is an important future direction.

Setup: The video server is hosted on a Linux machine (Ubuntu 16.04 OS, Intel i7-9700, 16 GB RAM, Radeon RX 550/550X GPU), and the video client runs on a Windows 10 machine (Intel i7-12700, 32 GB RAM, NVIDIA GeForce RTX 3070 GPU) as Oculus is only compatible with Windows. The network bandwidth between the server and client was emulated using Mahimahi [35]. The user watches the videos on an Oculus Quest 2 head mounted device (HMD) [9]. The HMD sends the user coordinates to the client machine every 40 ms, which in turn (i) sends the viewport back to a Unity App to render the frame on Oculus wirelessly using Air Link, and (ii) updates the future request of tiles to fetch it sends to the server. The client machine has enough computation resources to mask the rendering overhead induced by Unity app [12] and Oculus Airlink [8].

As discussed in §3.2, we have implemented two variants of Dragonfly with respect to transmission of the masking stream (an approach that transmitted the full 360°, and a tiling approach). Our user study uses the tiling approach. Specifically, it tracks the maximum displacement seen by past users (i.e., users in [34] dataset) on a per chunk basis for each video, and fetches masking tiles using this displacement value which varies across chunks. As shown by Figure 19 in the Appendix A, this scheme experiences slightly more incomplete frames, and slightly higher overhead than 360° masking. Thus, our user study understates the performance of Dragonfly.

Results. Figure 14(a) shows the distribution of the opinion score of all 26 participants across all their sessions (390 sessions in total, 130 for each system). Dragonfly is favored by the users with 65% of sessions having a rating of 4 and above. In comparison, only 16% of Pano and 13% of Flare sessions have this rating. Figure 14(b) shows the mean opinion score (MOS) of the users per video. Dragonfly has a higher MOS for all videos. Figure 14(c) shows the average PSNR for viewport tiles across all sessions of all users Dragonfly achieves a higher PSNR, with the median PSNR across viewports being 1.7dB higher than Pano, and 2.7dB more than Flare.

While Dragonfly may skip some tiles in the viewport, we found this was acceptable for two reasons. First, across sessions, the average percentage of incomplete viewports (viewports included one or more tiles that had at least one skipped frame) was small (1.13%). In contrast, the average rebuffering ratios across sessions with Pano and Flare were 3.35% and 2.77%, respectively. Further, skipped tiles were typically in the viewport periphery, where they have a lesser impact on perceived video quality. Figure 15 illustrates this by presenting a heat map that shows the fraction (across viewports of all sessions) that a tile was unavailable at a given location. The fraction was never higher than 0.8%, and the larger fractions were on the periphery.

Is user movement across systems comparable? In our study, participants are free to move as per their choice in each session. To verify that the better results were not owing to participants

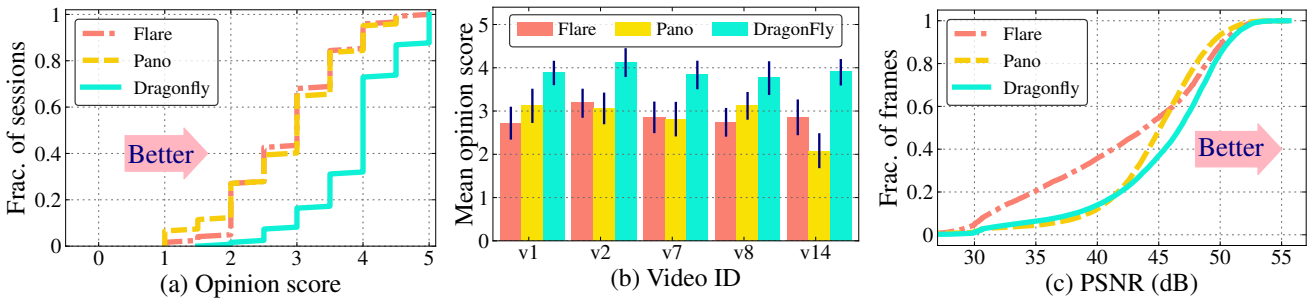


Figure 14: Results from user study. (a) Distribution of opinion scores across all users and sessions; (b) MOS per video along with 95% confidence intervals. Video IDs correspond to naming in [34]; and (c) Distribution of PSNR across users and sessions.

Figure 15: Dragonfly rarely experiences skipping; Skips mainly occur on the viewport edge where the impact is low.

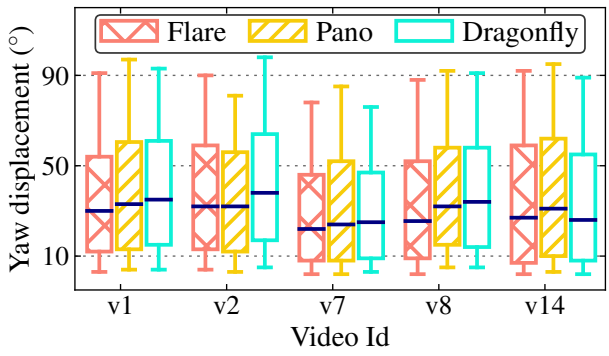


Figure 16: The distribution of users displacements during one second period across all sessions.

moving less with Dragonfly, Figure 16 shows the distribution of the yaw displacement estimated each second during all sessions. All systems experience similar yaw displacement across all videos indicating this was not a factor in the results. Interestingly, Flare saw slightly less displacement for video v8, and also had a lower MOS for the video. A potential explanation is that worse performance may reduce the user's interest in engaging with the video and consequently limits their movements – however, we defer an in-depth investigation to future work.

Qualitative feedback. Users were given the option to provide qualitative comments after they viewed each video with every system. Overall, we received 381 comments in our study. Since this is too numerous to enumerate, we summarize key observations from the qualitative feedback. We categorize each comment into: (i) blankness: the frequency of experiencing black tiles with a system; (ii) reactivity: the recovery time from low quality upon user movement; and (iii) quality: the perceptual video measurement. For instance, feedback such as "If quick flip, pixelated but improves. Response was fast" was indicative of a more reactive system, while feedback such as "It was taking forever to update", indicated a less reactive one. Likewise, "Extremely clear." would map to high quality, while comments such as "Details are not clear" or "Mostly pixelated, very bad" mapped to low quality.

Figure 17 summarizes participant feedback across all metrics. Figure 17(a) shows the degree to which different systems experienced blank screens according to the users. 90.15% (resp. 89.8%) of Pano (resp. Flare) feedback indicated the system experienced at least some or many blanks. In contrast, the corresponding number for Dragonfly was 47.1% – further, only 2.7% of the comments indicated Dragonfly experienced many blanks. Note that while Dragonfly experiences blanks when it skips tiles, Flare and Pano experience blanks when a rebuffering event occurs which may result in incomplete viewports rendered during the rebuffering event. Figure 17(b) shows the responsiveness of different systems to user motion. While the majority of feedback (73.7%) indicates Dragonfly is highly reactive, the majority of feedback related to Pano (57.2%) and Flare (78%) indicates the systems are slow to react. Finally, Figure 17(c) shows that 60.2% of user feedback indicates high perceptual quality with Dragonfly. In contrast, only 10% (resp. 6.74%) of feedback indicates high quality for Pano (resp. Flare).

5 RELATED WORK

§2 has discussed many tile-based streaming approaches [19, 21, 24, 31, 37–39, 42, 51] A recent approach [26] uses eye tracking to collect a user's gaze information, and uses live encoding to send uncompressed video frames for the foveal region and a compressed video frame stream for the rest. The paper focuses on the impact of latency on bandwidth savings of the approach, and shows that reducing end-to-end latency to 15 ms can save about 80% of bandwidth. In contrast, Dragonfly uses pre-encoded video chunks to reduce the latency like most other tile-based approaches. Recent work [46] uses dual queues and Deep Reinforcement Learning to guide fetch decisions DRL requires hours of training. Further, [46]

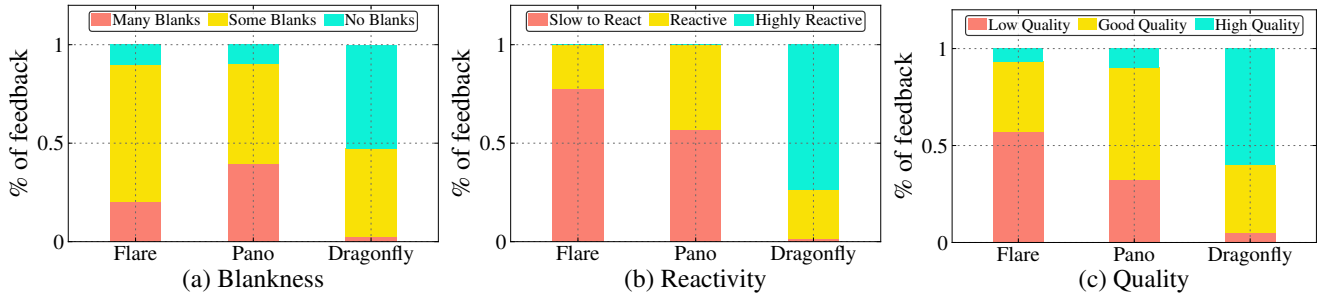


Figure 17: Participants feedback summary.

does not target continuous playback and reports noticeable stalls and modest bitrate improvements.

Like [24], some schemes stream the full 360° frame but use higher quality for the viewport. Rubiks uses more layers for the viewport [25], Core [36] encodes the viewport in higher resolution, and gradually reduces resolution at the periphery, while Meta researchers proposed encoding various viewport regions in the video at high quality (i.e., 150 different streams per video = 30 viewports x 5 quality levels), and only fetch the video stream with relevant viewport [29]. These approaches suffer from issues similar to Pano discussed in §2. Some works adapt the tiling scheme across a video session [50] or across the viewport in the frame [23] to reduce the bandwidth overhead and improve the QoE for the session. [23] assigns variable playback latencies for multiple users in a 360° video live streaming session. The viewport information from earlier users is used to predict the viewport for later users.

Some schemes [20, 22, 30] trade-off bandwidth requirements and client computation. [22] reduces bandwidth requirement for 360° video streaming using super resolution, while increasing computation at the client to infer high quality tiles from low resolution tiles. Other works [20, 30] are optimized for AR/VR workloads. Furion [30] fetches pre-rendered frames from the server for the less frequently changing background environment and uses local device rendering for foreground interactions. Alg-Vis [20] uses visual similarity of pixels across different VR frames to adaptively divide the frame into background and foreground. It renders the foreground and reuses the pixels to get the background.

6 CONCLUSION

In this paper, we have made three contributions. First, we have argued that in streaming 360° videos, it is preferable to skip tiles to preserve interactive experience rather than stall playback. Second, we have presented Dragonfly, a new 360° system that leverages the additional degrees of freedom provided by this design point. Dragonfly decides which tiles to fetch, and in what qualities, using a utility model that captures factors relevant to user experience. Further, Dragonfly tackles the degradation of prediction accuracy with look-ahead by fetching high quality tiles with a small look-ahead, and low quality versions with a longer look-ahead. Third, we have shown that Dragonfly outperforms state-of-the-art 360° streaming approaches. In our user study with 26 users, 65% of sessions have a rating of 4 or higher (Good/Excellent) with Dragonfly, while this is the case for only 16% and 13% of sessions with Pano and Flare, respectively. In our emulation experiments, Dragonfly achieved a median PSNR gain 1.72dB-4.5dB over existing approaches. None of Dragonfly sessions experienced incomplete viewports owning to its use of a masking stream. In contrast, at least one rebuffering event is observed for 99% of Flare sessions and 50% of Pano sessions. **This work does not raise any ethical issues.**

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback which greatly helped improve the paper. This work was funded in part by the National Science Foundation (NSF) Awards CNS 1956018 and 1956190, and KDDI Research, Inc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or KDDI.

REFERENCES

[1] 360 video on Vimeo. https://vimeo.com/channels/360vr.
[2] The 360° Effect | Understanding Immersive Video Research. https://magnaglobal.com/wp-content/uploads/2021/04/Magna.IPG-Lab-YuMe-The-360-Effect.pdf.
[3] End-to-end optimizations for dynamic streaming. https://engineering.fb.com/2017/02/22/virtual-reality/end-to-end-optimizations-for-dynamic-streaming/.
[4] How to watch Netflix VR? https://help.netflix.com/en/node/110502.
[5] Libavcodec Documentation. https://ffmpeg.org/libavcodec.html/.
[6] Libavformat Documentation. https://libav.org/documentation/doxygen/master/avio_8h.html/.
[7] Netflix recommended bandwidth. https://help.netflix.com/en/node/306.
[8] Oculus Airlink. https://www.oculus.com/blog/introducing-oculus-air-link-a-wireless-way-to-play-pc-vr-games-on-oculus-quest-2-plus-infinite-office-updates-support-for-120-hz-on-quest-2-and-more/.
[9] Oculus Quest 2. https://www.oculus.com/quest-2/.
[10] Oculus Virtual Rality. https://www.oculus.com/blog/what-is-virtual-reality-all-about/.
[11] Pano github. https://github.com/pkuguanyu/PanoProject.
[12] Unity. https://unity.com/.
[13] Vocabulary for performance and quality of service. https://www.itu.int/rec/T-REC-P.10.
[14] VQMT. https://github.com/rolinh/VQMT.
[15] Wyzowl. Survey: The State of Video Marketing 2018. https://www.wyzowl.com/video-marketing-statistics-2018/.
[16] YouTube recommended bitrates. https://support.google.com/youtube/answer/1722171?hl=en#zippy=%2Cvideo-resolution-and-aspect-ratio%2Cbitrate.
[17] YouTube VR. https://vr.youtube.com/.
[18] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 44–58, Budapest, Hungary, 2018.
[19] Y. Bao, H. Wu, T. Zhang, A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *IEEE International Conference on Big Data (Big Data)*, pages 1161–1170, Los Alamitos, CA, USA, 2016.

[20] Ying Chen, Hojung Kwon, Hazer Inaltekin, and Maria Gorlatova. Vr viewport pose model for quantifying and exploiting frame correlations. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, INFOCOM '22, page 1269–1278, London, United Kingdom, 2022.

[21] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *Proceedings of the Conference of the IEEE International Conference on Communications*, ICC '17, Paris, France, 2017.

[22] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R. Das. Streaming 360-degree videos using super-resolution. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, INFOCOM '20, pages 1977–1986, Toronto, ON, Canada, 2020.

[23] Amaya Dharmasiri, Chamara Kattadige, Vincent Zhang, and Kanchana Thilakarathna. Viewport-aware dynamic 360° video segment categorization. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '21', page 114–121, Istanbul, Turkey, 2021.

[24] 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*, SIGCOMM '19, page 394–407, Beijing, China, 2019.

[25] 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*, MobiSys '18, page 482–494, Munich, Germany, 2018.

[26] Luke Hsiao, Brooke Krajancich, Philip Levis, Gordon Wetzstein, and Keith Winstein. Towards retina-quality VR video streaming: 15ms could save you 80% of your bandwidth. *ACM SIGCOMM Computer Communication Review*, 52(1):10–19, 2022.

[27] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '14, pages 187–198, Chicago, Illinois, USA, 2014.

[28] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 97–108, Nice, France, 2012.

[29] Evgeny Kuzyakov and David Pio. Next-generation video encoding techniques for 360 video and VR. *Jan*, 21:3, 2016.

[30] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Transactions on Mobile Computing*, 19(7):1586–1602, 2020.

[31] Xing Liu, Qingyang Xiao, Vijay Gopalakrishnan, Bo Han, Feng Qian, and Matteo Varvello. 360° innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets '17, page 50–56, Palo Alto, CA, USA, 2017.

[32] 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*, VR/AR Network '17, page 30–35, Los Angeles, CA, USA, 2017.

[33] 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*, SIGCOMM '17, pages 197–210, Los Angeles, CA, USA, 2017.

[34] Afshin Taghavi Nasrabadi, Aliehsan Samiei, Anahita Mahzari, Ryan P. McMahan, Ravi Prakash, Mylène C. Q. Farias, and Marcelo M. Carvalho. A taxonomy and dataset for 360° videos. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, page 273–278, Amherst, Massachusetts, 2019.

[35] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Proceedings of the USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, page 417–429, Santa Clara, CA, 2015.

[36] Mijanur Palash, Voicu Popescu, Amit Sheoran, and Sonia Fahmy. Robust 360° video streaming via non-linear sampling. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, INFOCOM '21, pages 1–10, Vancouver, BC, Canada, 2021.

[37] 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*, MM '17, page 306–314, Mountain View, California, USA, 2017.

[38] 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*, MobiCom '18, page 99–114, New Delhi, India, 2018.

[39] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ATC '16, page 1–6, New York City, New York, 2016.

[40] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. Beyond throughput, the next generation: A 5g dataset with channel and context metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 303–308, Istanbul, Turkey, 2020.

[41] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, INFOCOM '16, pages 1–9, San Francisco, CA, USA, 2016.

[42] Kashyap Kammachi Sreedhar, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications. In *Proceedings of the IEEE International Symposium on Multimedia*, ISM '16, pages 583–586, San Jose, CA, USA, 2016.

[43] Liyang Sun, Fanyi Duanmu, Yong Liu, Yao Wang, Yinghua Ye, Hang Shi, and David Dai. A two-tier system for on-demand streaming of 360 degree video over dynamic networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):43–57, 2019.

[44] Guibin Tian and Yong Liu. Towards agile and smooth video adaptation in dynamic http streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, Nice, France, 2012.

[45] 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.

[46] Chenglei Wu, Zhi Wang, and Lifeng Sun. Paas: A preference-aware deep reinforcement learning approach for 360° video streaming. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '21, page 34–41, Istanbul, Turkey, 2021.

[47] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '20, pages 495–511, Santa Clara, CA, 2020.

[48] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *Proceedings of the USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, pages 731–743, Boston, MA, 2018.

[49] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 325–338, London, United Kingdom, 2015.

[50] Lei Zhang, Yanyan Suo, Ximing Wu, Feng Wang, Yuchi Chen, Laizhong Cui, Jiangchuan Liu, and Zhong Ming. Tbra: Tiling and bitrate adaptation for mobile 360-degree video streaming. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 4007–4015, Virtual Event, China, 2021.

[51] Chao Zhou, Mengbai Xiao, and Yao Liu. Clustile: Toward minimizing bandwidth in 360-degree video streaming. In *Proceedings of the Conference of the IEEE Computer and Communications Societies*, INFOCOM '18, pages 962–970, Honolulu, HI, USA, 2018.

A APPENDIX

Appendices are supporting material that has not been peer reviewed.

Algorithm 1 Utility Algorithm (§3.1)

```
req = list();  // tiles request
U_sum = 0;  // This is the overall utility
ζ ← C  // ζ contains the set of all tiles in C
for q : 1 → Q do
    δ ← sort i by U_{i q0 t0}; ∀i ∈ ζ  // q0 is the current quality
    for i ∈ δ do
        idx ←  tentative position of i in Req s.t. max U_sum
        if idx ==  null then
            continue to next tile;
        req[idx] ← i  // add i to request at position idx
        q0[i] ← q  // upgrade quality of i to q
        for tile k ∈  subsequent(i)  do  // k_idx > idx
            while U_{k,q0,t} == 0 do  // if k utility is zero
                q0[k] ← q0[k] − 1  // downgrade k quality.
                 // reduce k size, thus the arrival time
                Update t[k]  // update k arrival time.
                 // This can increase k utility > 0
            // if k size reduction did not help
            if U_{k,q0,t} = 0 then
                Drop  k  from  req
    // update the set tiles ζ to only tiles in the req.
    ζ ← req
```

Sensitivity to quality (§3.1). Figure 18 shows an example of two tiles from the same video encoded at the highest and lowest quality levels. For the bottom tile, a higher quality (right) is perceptually much better than a lower quality (left). For the top tile, the difference is not as sharp.

(a) low perceptual difference

(b) high perceptual difference

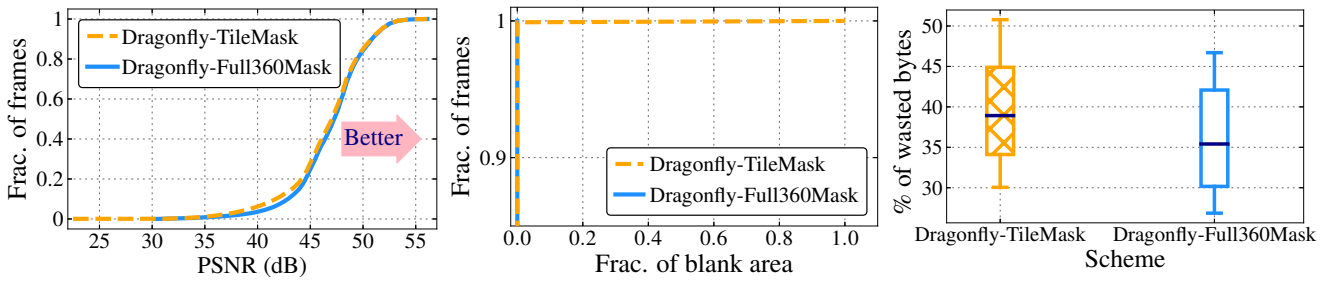
Figure 18: Example of two tiles at the lowest and highest quality levels.

Comparing masking strategies (§3.2,§4.5). We compare two different masking strategies that we have implemented for Dragonfly: masking using the full 360°, and a tile masking scheme. For tile masking, the maximum displacement around the user's coordinate that the masking stream must fetch is a configurable parameter that can be specified on a per chunk basis. We set this based on the maximum displacement seen over a subset (20) user trajectories in [34] dataset on a per chunk per video basis, and evaluate the scheme on the remaining (10) user trajectories. Figure 19 shows the two variants perform comparably, although tile masking experiences slightly more incomplete frames and overhead. The higher overhead is because for low qualities (that the masking stream is transmitted in), encoding is more efficient when the full 360° is transmitted compared to a tiling approach even though this corresponds to a larger region.

Why 12x12 tiling? (§4.2) Our evaluations use a 12x12 tiling. Using larger tiles (i.e., partitioning into less than 144 tiles) potentially leads to better compression (see discussion related to 4.4.1 below), while having smaller tiles helps in minimizing the data to be transmitted if the user moves frequently (over a chunk duration of 1 second, any tile that overlaps with any viewport frame in that 1 second period must be transmitted). We did trace-driven simulations to derive the video bit rate needed if the viewports were perfectly predicted and only these viewports were streamed. Our simulations varied tile size, and used user trajectories from our data-set. For a higher quality encoding, our results show that a user with a 12x12 tiling scheme would need 5.45% less bandwidth compared with a finer 24x18 tiling scheme and 20% less bandwidth than a coarser 6x6 tiling scheme.

Compression benefits of using Pano's variable tiling (§4.3) Pano splits a chunk into 30 variably sized groups of tiles (so that a given tile contains pixels with a similar quality sensitivity to changes in encoding parameters), while Dragonfly (and other systems) use fixed tiling. In our implementation, this involves dividing each chunk into 144 (12x12) equally sized tiles. In Figure 20, we compare the sizes of videos used in our evaluations (across different qualities) for the two tiling schemes. The X-Axis shows the size of videos encoded using Pano's tiling scheme, and the Y-Axis shows the ratio of the overheads due to fixed tiling (i.e., $\frac{F}{V}$, where F is the size with fixed tiling, and V the size with variable tiling). Although there is noticeable overhead for lower quality videos (consistent with [24]), the overheads degrade significantly at larger quality levels and higher bit rate videos. The main reason why variable tile sizes helps is because of better opportunities for intra-frame prediction. However, intra prediction helps primarily at low rates. At high rates the difference between intra prediction and blocks encoded directly without prediction becomes negligible. For 360° streaming with HMDs, the bitrates tend to be high with 4K and higher resolution, hence we believe the benefits of variable tiling will in general be lower. However, the additional costs must still be incurred (as regions outside the viewport may be part of the viewport tile).

Sensitivity to motion prediction errors (§4.3). To verify the robustness of Dragonfly to errors in motion predictions, we evaluate Dragonfly and other schemes, over 7 videos, 5 users traces, and



(a) Distribution of PSNR across viewports of all sessions. (b) Frac of blank pixels per viewport. (c) Percentage of wasted bytes.

Figure 19: Objective quality metrics comparison of Dragonfly variants which use either full 360° for masking (emulation) or tile based strategy (user study).

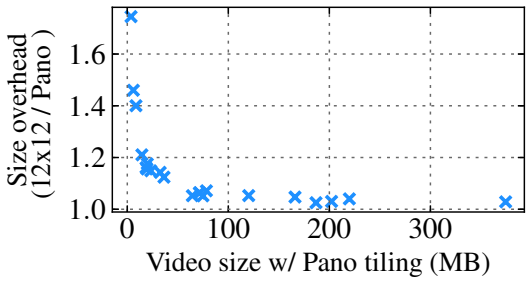


Figure 20: Comparing the encoding overhead using 12x12 tiling scheme and Pano grouped tiling scheme.

Video ID	median QP42 (Mbps)	median QP22 (Mbps)
v1	0.9	10.4
v2	1.2	10.5
v7	1.7	24.4
v8	3.1	28.4
v14	3.3	27.8
v28	3.6	30.9
v27	4.6	49.6

Table 3: Summarizing median videos bitrates for the lowest (QP42) and highest (QP22) qualities (sorted by QP42).

5 bandwidth traces from Belgian dataset, by shifting the history of viewport coordinates by random degrees D which is uniformly distributed, following previous work [24]. Results (Figures 21-23 in Appendix §A) shows that Dragonfly achieves higher PSNR for different D relative to other schemes, while maintaining high interactive experience with only 1% of sessions encounter incomplete viewports. Looking at the overheads, the trend persists with Dragonfly having the lowest median overhead.

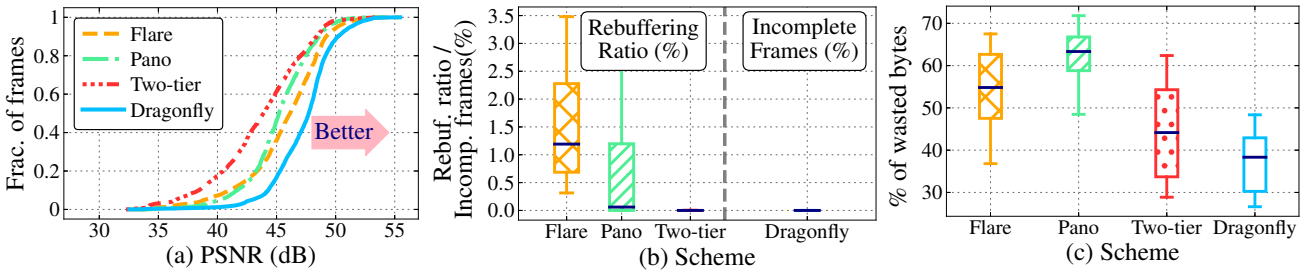


Figure 21: Objective quality metrics. Viewport coordinates shifted by D = 5 degrees.

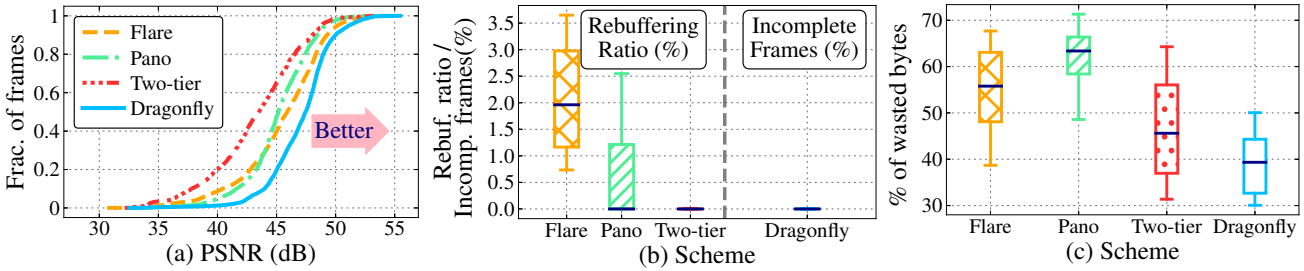


Figure 22: Objective quality metrics. Viewport coordinates shifted by D = 20 degrees.

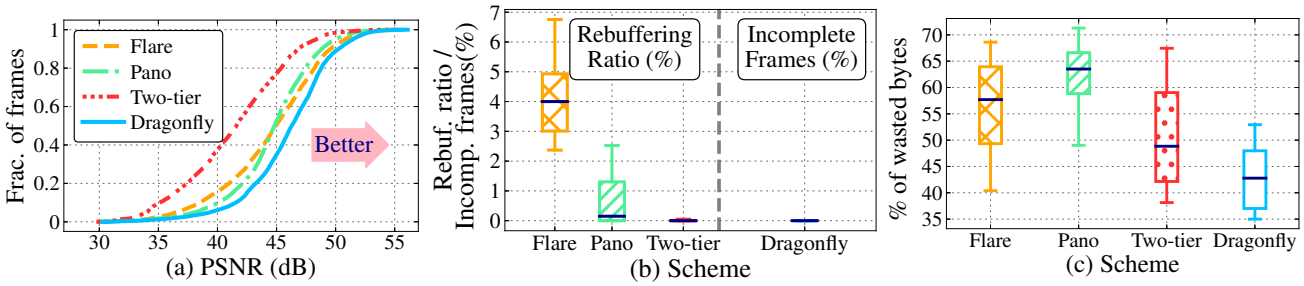
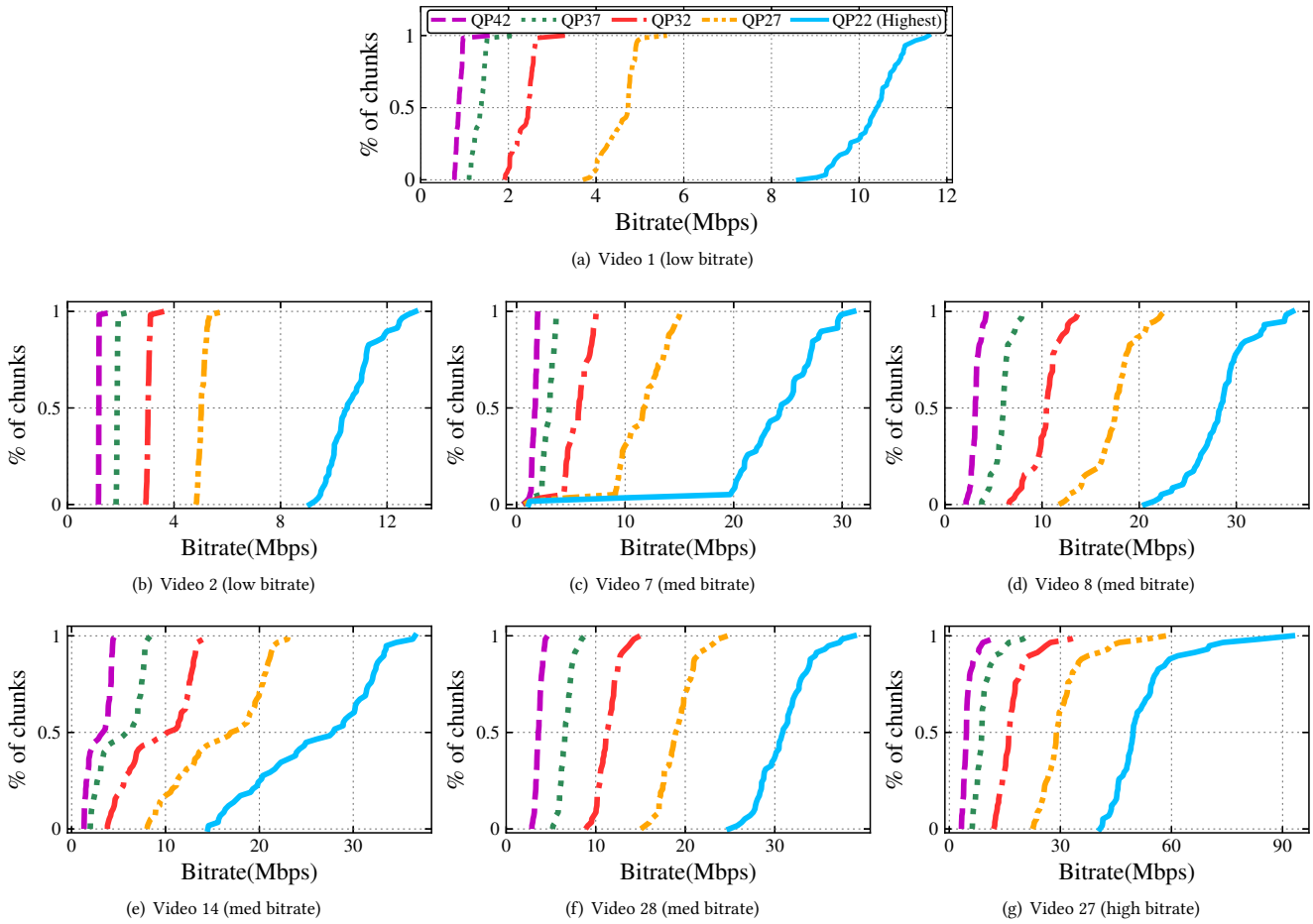


Figure 23: Objective quality metrics. Viewport coordinates shifted by D = 40 degrees.



(a) Video 1 (low bitrate)

(b) Video 2 (low bitrate)

(c) Video 7 (med bitrate)

(d) Video 8 (med bitrate)

(e) Video 14 (med bitrate)

(f) Video 28 (med bitrate)

(g) Video 27 (high bitrate)

Figure 24: The bitrate of full 360° chunk: The videos bitrate for all qualities, 2 videos [a-b] are low bitrate, 4 videos [c-f] are medium bitrate videos, and one high bitrate video[g].