REEFT-360: Real-time Emulation and Evaluation Framework for Tile-based 360° Streaming under Time-varying Conditions

Eric Lindskog
Linköping University, Sweden

Niklas Carlsson
Linköping University, Sweden

ABSTRACT

With 360° video streaming, the user's field of view (a.k.a. *viewport*) is at all times determined by the user's current viewing direction. Since any two users are unlikely to look in the exact same direction as each other throughout the viewing of a video, the frame-by-frame video sequence displayed during a playback session is typically unique. This complicates the direct comparison of the perceived Quality of Experience (QoE) using popular metrics such as the Multiscale-Structural Similarity (MS-SSIM). Furthermore, there is an absence of light-weight emulation frameworks for tiled-based 360° video streaming that allow easy testing of different algorithm designs and tile sizes. To address these challenges, we present REEFT-360, which consists of (1) a real-time emulation framework that captures tile-quality adaptation under time-varying bandwidth conditions and (2) a multi-step evaluation process that allows the calculation of MS-SSIM scores and other frame-based metrics, while accounting for the user's head movements. Importantly, the framework allows speedy implementation and testing of alternative head-movement prediction and tile-based prefetching solutions, allows testing under a wide range of network conditions, and can be used either with a human user or head-movement traces. The developed software tool is shared with the paper. We also present proof-of-concept evaluation results that highlight the importance of including a human subject in the evaluation.

CCS CONCEPTS

• **Information systems → Multimedia streaming**; • **Human-centered computing → Interactive systems and tools**; • **Networks → Application layer protocols**.

KEYWORDS

360° video, Tile-based streaming, Real-time emulation, Evaluation framework, Time-varying, Head movements, Bandwidth variations, Oculus, Unity

ACM Reference Format:
Eric Lindskog and Niklas Carlsson. 2021. REEFT-360: Real-time Emulation and Evaluation Framework for Tile-based 360° Streaming under Time-varying Conditions. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3458305.3478453

1 INTRODUCTION

The emergence of 360° streaming comes with both great opportunities and new challenges. Compared with traditional video streaming, most new challenges come from the observations that (i) not all directions are equally important and (ii) the difficulty to reliably predict users' head movements [4, 9, 19]. For example, with 360° video only a limited fraction of the full view (called the *viewport*) is displayed at each moment in time. To reduce the bandwidth usage and/or to improve the expected playback quality given the available bandwidth, much research has therefore focused on the design of techniques that predict head movements and deliver different video quality for each potential viewing direction [4, 9, 12, 16, 24]. This includes the development of tile-based solutions, in which (i) the full 360° view is split into several tiles, (ii) each tile is encoded at different qualities, and (iii) the clients can prefetch and display tiles of different qualities for each direction.

By adapting the quality selection of each tile based on *both* the current network conditions and the expected viewing direction, the users can then try to maximize their expected Quality of Experience (QoE). However, the users' added freedom to select their own viewing direction at each moment in time presents unique *evaluation challenges* compared to the context of traditional linear Dynamic Adaptive Streaming over HTTP (DASH) [25], where all users watch the same frame-by-frame sequence. For example, with 360° streaming, the viewport always depend on the user's current viewing direction. Two arbitrary users watching the same video are therefore highly unlikely to observe exactly the same frame-by-frame video sequences and playback sessions are typically unique. This complicates the direct comparisons of users' QoE using objective, visual, frame-based metrics such as Multiscale-Structural Similarity (MS-SSIM) [27] and Peak Signal-to-Noise Ratio (PSNR). Motivated by the value that such metrics have provided in the context of regular linear video [7], we argue that it is important to provide evaluation frameworks that allow these metrics to be easily calculated for 360° streaming too. Today, no such framework exist.

In this paper, we address the above *evaluation challenges* and void through the development of REEFT-360 (first-five capped letters in paper title + 360): a novel *real-time emulation framework* and a *multi-step evaluation process* that allow us to calculate MS-SSIM and other frame-based QoE metrics while accounting for the user's head-movements when watching the tiled video over emulated networks with time-varying bandwidth conditions. REEFT-360 is the first of its kind in that it allows such traditional frame-based QoE metrics to be calculated for 360° streaming. Our light-weight emulation framework fills an important gap, and enables easy testing of different algorithm designs, conditions, and tile sizes.

The *emulation framework* incorporates a modular design that easily can emulate trace-based bandwidth profiles capturing a broad range of network conditions, bandwidth manipulations by network operators, the use of alternative tile sizes, chunk durations, and alternative head-movement prediction algorithms and prefetching solutions. For speedy implementation and testing, we have implemented a GPU-based solution that emulate the viewport shown to the users in real-time on a frame-by-frame basis using only the original high-quality 360° frames available locally on the machine, meta file information (e.g., tile sizes, chunk duration, encoding levels, etc.) that the user can change very quickly, and information about what file data have been obtained in time of playback. The *multi-step evaluation process* can be used both with a human user (suggested for good evaluation) and head-movement traces (for speedy volume testing) collected with this or other frameworks. In both cases, it allows easy calculation of MS-SSIM scores, PSNR, and other frame-based QoE metrics.

To demonstrate how the tool can be used, we present a simple proof-of-concept evaluation that highlights the importance of including a human subject in the evaluation of tile-based prefetching algorithms. Drawing comparisons to control theory, we note that putting a *human in the loop* can impact the best adaption approaches. For example, in addition to head movements impacting the QoE, the example results also suggest that the tile qualities being presented themselves impacts the head movements. In fact, in some scenarios we observe that the head movements can be greatly restrained by the degraded tile qualities that a user sees (e.g., in the periphery of the viewport) when too aggressive prefetching algorithms are used or the network conditions are poor. This further highlights the value of the tool presented and shared with this paper, which should help others to easily implement and evaluate their own tile-based 360° solutions with a human in the loop. As of today, most prior evaluation of such prefetching solutions ignores this aspect.

Code, software, and example datasets can be found on Github (https://github.com/EricLindskog/REEFT-360) with doi (https://www.doi.org/10.5281/zenodo.5155943).

Outline: The remainder of this paper is organized as follows. Section 2 presents the design and implementation of REEFT-360. Section 3 presents an example evaluation. Finally, we discuss related work (Section 4) and present conclusions (Section 5).

2 DESIGN AND IMPLEMENTATION

REEFT-360 consists of a real-time emulation framework (Section 2.1) and a multi-step evaluation process (Section 2.2).

2.1 Design of emulation framework

Figure 1 shows a high-level overview of the emulation framework.

Player logic: At the center of the design is the `Player` component. It implements all playback logic and emulates what happens to each chunk when inside the player, including what happens to the buffer when new data is obtained over the network and what happens with file data when the user pauses/resumes playback. It also ensures that the video stalls whenever the buffer goes empty.

Bandwidth estimation and quality selection: To simplify the comparison of different bandwidth estimation and quality-selection techniques, the player delegates the logic for these aspects to a `Bandwidth Estimator` and chunk `Requester` component, respectively. `Requester` decides when and what chunk tiles to request. Here, we use the term *chunk tile* to refer to the individual tile

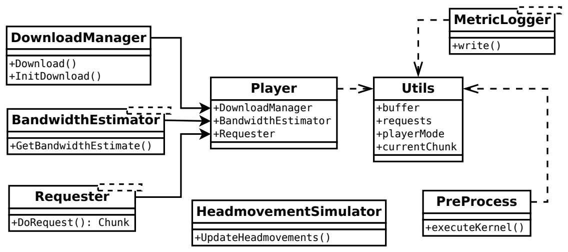


Figure 1: High-level design of emulation framework

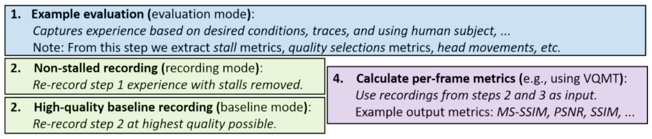
of a chunk. For these decisions the `Requester` component has access to bandwidth estimation by `Bandwidth Estimator` and global variables regarding the player state. A custom class called `Utils` is responsible for keeping track of global states. This class does not perform any actions but instead provides a central access point to variables (e.g., chunk sizes, buffer states, the current player mode, etc.) and functions used by several other components. The use of a single shared interface simplifies adding shared variables and makes sure that all functions access the variables/functionalities the same way.

Download processes and network aspects: After a chunk quality selection has been made, the `Player` calls a class called the `Download Manager`. This class (i) uses bandwidth traces and network logic (e.g., apply bandwidth caps) to simulate the actual network conditions that the client would observe, (ii) determines when the file download of each chunk tile completes, and (iii) informs the player of chunk-tile download completions.

Logging and viewport visualization: We use independently run processes for several purposes, including a `Metric Logger` that continuously logs the current player state and a `PreProcess` component that emulates how lower-quality tiles are displayed. To allow easy replaying of a session, the `Metric Logger` records head movements (e.g., pitch, yaw, roll), the chunk-tile qualities requested, and a video of the session as viewed through the viewport. In addition, we record a wide range of performance metrics, including the stalls (e.g., start time + duration), the tile in focus, and the quality levels of the tiles shown within the viewport at each moment.

For increased flexibility and faster comparison of alternative candidate solutions, the `PreProcess` component emulates the results from pre-processing steps such a chunking, tiling, and general video encoding at run time. (Section 2.3 presents our GPU-based implementation.) As input, it takes a meta file specifying the video segments and tiles, and then, at run time, it renders an estimation of the viewport shown to a client that obtain that specific set of chunks. In this step, we downscale the quality of the region of the viewport (from a locally stored high-quality copy) based on the quality that the client was emulated to have obtained for each tile.

Player modes: Our system can be run in four different modes: *Testing*, *Evaluation*, *Recording*, and *Baseline*. The first two modes differ only based on whether traces are recorded (*Evaluation*) or not (*Testing*). Both these modes leverage the full power of the emulation framework and can either be ran by (1) a human wearing a Head Mounted Display (HMD), or (2) using pre-recorded head-movement traces. When using head-movement traces, an independent component (`HeadmovementSimulator`) is responsible for emulating the



1. Example evaluation (evaluation mode**):**
Captures experience based on desired conditions, traces, and using human subject, ...
Note: From this step we extract *stall* metrics, *quality selections* metrics, *head movements, etc.*

2. Non-stalled recording (recording mode**):**
Re-record step 1 experience with stalls removed.

2. High-quality baseline recording (baseline mode**):**
Re-record step 2 at highest quality possible.

4. Calculate per-frame metrics (e.g., using VQMT**):**
Use recordings from steps 2 and 3 as input.
Example output metrics: *MS-SSIM, PSNR, SSIM, ...*

Figure 2: Multi-step evaluation methodology overview

head movements. In the *Recording* mode, a prior *Evaluation* experiment is re-recorded using the same head movements and quality selections but with all stalls removed. Finally, in *Baseline* mode, a prior *Recording* experiment (without stalls) is re-recorded with all tiles at the highest quality but exactly the same head movements.

2.2 Multi-step evaluation methodology

To calculate MS-SSIM, PSNR, and other frame-based quality metrics, we implement a four-step evaluation methodology. See Figure 2.

1. Example evaluation: In the first step, we use the *Evaluation* mode to capture the experience of (i) a test subject wearing an HMD while interacting with the video or (ii) a client who's head movements follow a particular trace file. Here, the network is emulated based on one or more network traces and information specifying a potential bandwidth cap. For each test, a specific chunk-tile selection technique and bandwidth estimation technique must be specified. During the test, we then record information about playback quality, stalls, tile selection, and head movements.

2. Non-stalled recording: For the second step, we use the *Recording* mode to record the exact viewport seen by the client from the first step, but with any stalls removed. This playback session emulation is produced using the playback quality and head movement traces from the first step.

3. High-quality baseline recording: For the third step, we use the *Baseline* mode to record the viewport of a client with identical head movement as used for step 2 but when each tile is obtained and watched at the highest possible quality.

4. MS-SSIM calculations: Since the recordings from steps 2 and 3 (due to stall removals) always capture the same video direction and playpoint at each moment in time, the two recordings can be compared frame-by-frame. This allows MS-SSIM, PSNR, and other per-frame-based video quality metrics to be easily calculated. Here, we use the *Unity Recorder*[1] for video recordings and the video quality measurement tool (VQMT)[2] by Wang et al. [27] to extract the MS-SSIM, PSNR, and other per-frame scores for every frame.

2.3 Emulation framework implementation

Our framework is implemented in Unity using C♯[3]. For much of the implementation, we extend the `MonoBehaviour` class[4] and utilize some of its functions, including `Start()`, `Update()` and `FixedUpdate()`. When applied to a game component, a script extending `MonoBehaviour` uses the `Start()` function to create an instance of the game component and the `Update()` function is called once every frame. In contrast, the `FixedUpdate()` function is called every t time units (fixed time interval), independent of the applications frame rate, making it ideal for logging.

Light-weight simulation of lower quality tiles: To allow emulation at run-time and eliminate pre-processing (e.g., to encode multiple video versions and tile sizes), the `PreProcess` component implements two methods: (i) *lowering resolution* and (ii) *compression emulation*. Both methods are implemented on a per-frame basis and are called upon using the `Update()` function.

The *lowering resolution* function modifies the displayed video frame-by-frame in real-time. Since this calculation would not be feasible on a CPU, this function was implemented using the compute shader framework available in unity, which runs the code on the GPU. To lower the resolution we simply group pixels and use their Red Green Blue Alpha (RGBA) values; effectively lowering the resolution. For example, if the source is a 4K video, then averaging over two-by-two blocks will result in output at approximately 1080p, depending on the exact dimensions of the video. The number of pixels grouped are specified using the *block size*.

We also implement *compression emulation* at run-time using a compute shader. The compression effect is created at run-time by calculating the difference between the RGBA value of each pixel of the previous frame and the pixels of the upcoming one, only updating it if the difference is above a certain threshold. These comparisons are only made in a set interval of frames after which the full image is updated. The compression allows for more granular control of playback quality than simply altering the resolution; e.g., by supplying the two threshold parameters for when to update a pixel and the interval at which the entire image is updated.

While more advanced compression and encoding algorithms than those emulated here often are used in practice, such solutions are typically system dependent. Here, we instead aim for a basic and easily extendable framework that provides easy and fair head-to-head comparisons of different chunk-tile selection and head-movement prediction algorithms.

Network components: The download progress of each chunk tile is emulated by the `Download Manager`. This component uses one or more trace files and other information (e.g., bandwidth caps) to determine the download speed at each time instance. This is achieved by considering one trace file entry at a time and maintaining state information about each ongoing download. In the case that a bandwidth cap C is used, the minimum value of the cap and the trace value is used as the bandwidth obtained by the client.

2.4 Chunk-tile selection algorithms

Chunk-tile selection and bandwidth estimation techniques can easily be incorporated via the `Requester` and `Bandwidth Estimator` component, respectively. For our proof-of-concept evaluation, bandwidth estimates are calculated using an exponentially weighted moving average (EWMA) with weight α=0.3 given to the most recent sample. We next describe two chunk-tile selection algorithms.

DASH baseline: The first algorithm serves as a baseline. It implements a traditional DASH-like algorithm that ignores the viewing direction. With this algorithm, all tiles of a chunk are requested at the same quality. Given this constraint, it greedily selects the highest quality allowed by the current bandwidth estimate. In particular, we pick quality j^* such that $j^* = \arg\max_j \left[\sum_i q_{i,j} \mid \sum_i q_{i,j} < B\right]$,

[1]https://docs.unity3d.com/Packages/com.unity.recorder@2.0/manual/index.html
[2]https://www.epfl.ch/labs/mmspg/downloads/vqmt/
[3]https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html
[4]https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Algorithm 1 Waterfill (Definitions of $\Delta^q_{i,j(i)}$ and $\Delta^u_{i,j(i)}$ in text.)

```
1: $j(i) \leftarrow 0, \forall i$
2: while $(\sum_i q_{i,j(i)} + \min_i \ \Delta^q_{i,j(i)}) \leq C$ do
3:     $i' \leftarrow \arg\max_i [w_i(\Delta^u_{i,j(i)}/\Delta^q_i) \mid \sum_i q_{i,j(i)} + \Delta^q_{i,j(i)} \leq C]$
4:     $j(i) \leftarrow j(i) + 1$
5: end while
```

Table 1: Quality validation using reference video

Resolution	Block size	MS-SSIM (real)	MS-SSIM (emulate)	Bitrate (kbps)
4k	1	1.0000	1.0000	35000
1080p	2	0.9765	0.9854	8000
720p	4	0.9483	0.9498	5000
480p	6	0.9186	0.9157	2500
None	-	-	0	0

where $q_{i,j}$ is the bandwidth cost of downloading tile i of the chunk at quality level j, and B is the estimated available bandwidth.

Waterfill algorithm: The second example algorithm is used to illustrate the tradeoffs associated with prioritizing tiles towards the center of the expected viewport. To understand its rationale, we first note that the quality selection problem can be formulated as a packing problem that optimizes the weighted sum of the downloaded tile qualities $\sum_i w_i q_{i,j(i)}$, where $j(i)$ is the level choice made for tile i, given the total bandwidth constraint $\sum_i q_{i,j(i)} \leq B$. Second, we note that finding the optimal solution is of exponential complexity. Instead, we design a light-weight heuristic.

Algorithm 1 presents our *waterfill* algorithm, which is an adaption of a similar heuristic from the stream bundle context [5]. Starting with the null-assignment (line 1), in each step, we greedily (lines 3+4) identify the tile that maximizes the weighted gain in utility $w_i\Delta^u_{i,j(i)}$ (if upgrading tile i one quality level) relative to the additional bandwidth this would consume (i.e., $\Delta^q_{i,j(i)} = (q_{i,j(i)+1} - q_{i,j(i)})$), conditioned on not exceeding the estimated available bandwidth C. Here, we assume that the change in utility is $\Delta^u_{i,j(i)} = \Delta^q_{i,j(i)}$ whenever $j(i) > 0$ and $\Delta^u_{i,0} = q_{i,1} + A$ otherwise. The parameter A gives weight to the importance of avoiding missing tiles. For our experiments, weights w_i are assigned using a simple heuristic $w_i = \Delta^{lat}_i/180$, where Δ^{lat}_i is the latitude distance between the user's viewing direction and the center of tile i.

2.5 Validation of block-size approach

To get a reference point for how well our block-sized emulation approach for adapting the resolution in real-time captures the quality reduction of each frame, we acquired the *Big Buck Bunny* video in 4k, re-encoded it to 1080p, 720p and 480p, and used the VQMT framework to calculate the average MS-SSIM values for the different encodings. (The bitrates values were selected based on YouTube recommendations.[5]) These MS-SSIM values were then compared to when using the basic block-sized approach used here. The results are shown in Table 1, and shows that our basic approach nicely captures the general bitrate-QoE tradeoffs.

[5]https://support.google.com/youtube/answer/1722171

3 EXAMPLE EVALUATION

To illustrate how REEFT-360 can help better understand system tradeoffs, we next present a performance evaluation. Here, we show results when using example videos, prefetching algorithms, and buffer sizes under different network conditions and both with and without bandwidth caps. All experiments were done by a single example user and we use MS-SSIM to illustrate the use of a frame-based metric enabled by our framework.

3.1 System setup

Hardware: The experiments were performed on a computer running Windows 10 equipped with an AMD Ryzen 2700 CPU, NVIDIA GTX 1070 GPU and 16 gigabytes of RAM. Oculus Rift CV1 was used as HMD. This setup ensured that we were not resource constrained. For example, during our experiments, the utilization typically was around 25% (CPU), 30% (GPU) and 10-15% (RAM).

Bandwidth traces: We used public bandwidth traces [20] to drive our network emulation framework. For the results presented here, we used 400+ seconds of the trace *static/A_2017.11.30_16.15.00* to capture the download bandwidth variations and multiplied each such data point with a factor β to emulate networks with different average bandwidth conditions. To emulate scenarios with *poor*, *fair*, and *good* network conditions we used β values of 1.25, 2.5, and 5. This resulted in average bandwidths of 2,500 kbps, 5,000 kbps, and 10,000 kbps, respectively. For the *capped* emulations we used $\beta = 5$ but capped the bandwidth at 6,000 kbps.

Videos: We used the two videos that Almqvist et al. [2] previously selected as representative of a typical *rides* and *exploration* video, respectively. Both videos use an equirectangular projection. For the experiments, both videos run for 150 seconds (the duration of the shorter video), and we used eight horizontally distributed tiles with a chunk duration of one second. The choice to not use vertical tiling speeds up the implementation of the more advanced fetching algorithm and assures they can run in real-time.

Experiments: All experiments followed our multi-step methodology (Section 2.2) and sufficient breaks were taken to ensure that simulator sickness was not an issue [23]. At the start of each experiment, we set all emulation parameters and scaled the bandwidth traces. All *Evaluation* mode experiments were conducted with the test subject wearing the HMD, before *Recording* and *Baseline* mode recordings were generated and used for the MS-SSIM calculations.

3.2 Example MS-SSIM results

Consider first a *fair* bandwidth scenario when watching the *exploration* video using the *waterfall* algorithm with A=200.

Figure 3(a) shows frame-by-frame MS-SSIM values for such a session. Here, the three colored lines corresponds to the reference values for low (red), medium (orange) and high (green) that we determined for the reference video (Table 1). We note several cases when the frame quality degrades substantially. These instances typically correspond to frames where some tile is missing from the user's viewport. This observation is reflected in the high correlation between these frames and the frames marked in red in Figure 3(b). Here, we show the average quality of each frame as (naively) approximated by the average quality level calculated over the tiles visible in each frame (with quality levels labeled 1-through-4 and

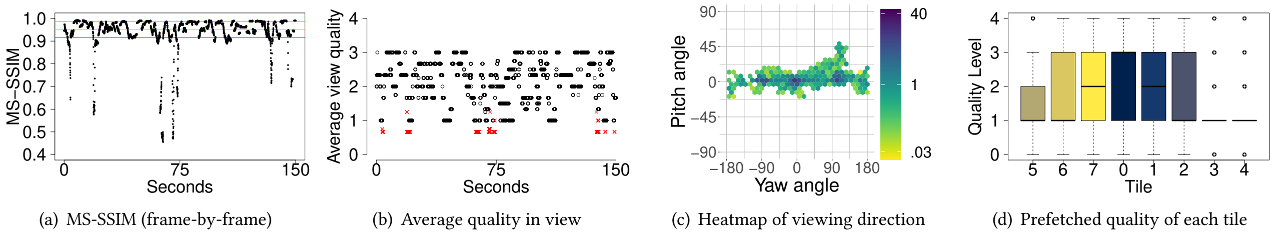


Figure 3: Frame-by-frame MS-SSIM values, average qualities, and other example results. Here, the user watched the *exploration* video under the *fair* bandwidth scenario and the client implemented Waterfall with $A = 200$.

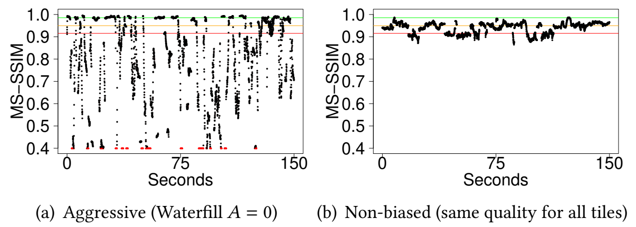


Figure 4: Comparing two extreme policies: MS-SSIM values for *exploration* video and *fair* bandwidth scenario.

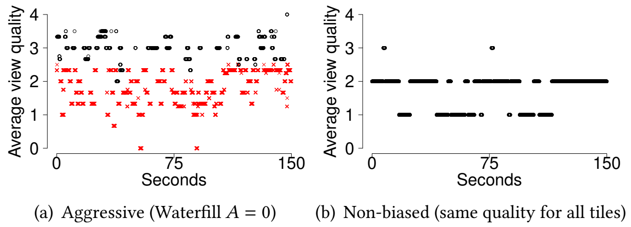


Figure 5: Comparing the two extreme policies: Average quality for *exploration* video and *fair* bandwidth scenario.

level 0 corresponding to a missing tile) and mark frames with one or more tiles missing from the viewport using red color. However, while the average metric (Figure 3(b)) provides a simple proxy to the MS-SSIM (Figure 3(a)), it misses several instances. This highlights the need for more advanced evaluation frameworks, such as REEFT-360.

Figure 3(c) shows the head movements as a heat map of the yaw and pitch angle. Note that the head movement in this *exploration* scenario can be substantial. A more aggressive prefetching policy that down-prioritize tiles in currently non-viewed directions is therefore likely to see some missing tiles. Figure 3(d) shows box-whisker plot of the tile qualities displayed to the user. Here, tile 0 represents the tile straight ahead and the tiles are numbered clockwise. As per design, the highest quality chunks are obtained in the center direction, but the algorithm typically also obtains at least quality level 1 in most other directions. With a more *aggressive* prefetch algorithm (*waterfill* with $A = 0$) we observed much higher degradation for this scenario (e.g., MS-SSIM values in Figure 4(a) and average quality in Figure 5(a)). In contrast, the basic DASH policy (that selects the same quality in all directions) observed more stable frame-by-frame qualities (Figures 4(b) and 5(b)).

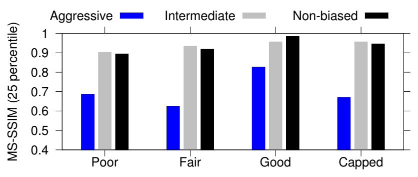
3.3 Performance comparisons

We next illustrate how the tool can be used to study effects that are not possible to study without including a human subject in the evaluation. Here, we present a scenario-based evaluation that includes results for the four bandwidth scenarios defined in Section 3.1 (*poor*, *fair*, *good*, and *capped*) when using prefetching algorithms that are more or less aggressive: *aggressive* (waterfill with $A = 0$), *intermediate* (waterfill with $A = 200$), and *non-biased* (basic DASH with same quality in all directions). Figure 6 shows example results using three summary metrics: (1) the 25th percentile of the MS-SSIM scores, (2) the fraction of frames with MS-SSIM scores of at least the low-quality reference, and (3) the total head movement per session. In the discussion, we also provide statistics for the number of stalls, mean stall duration, and other example metrics.

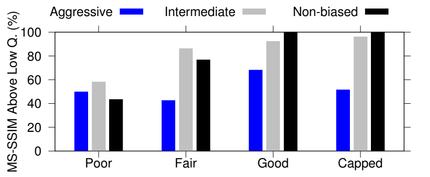
Video-type comparison: These results confirm that too aggressive prefetching is undesirable for both video types considered. For example, the *aggressive* prefetching policy (blue bars) consistently is outperformed by the *intermediate* policy (gray bars) and the non-biased policy (black). In addition to the lower MS-SSIM values (example metrics in first two columns in Figure 6) we also note that too aggressive prefetching significantly reduces the user's head movements (third column). We expect that this behavior is due to a combination of psychological effects (e.g., the user is less likely to move their head to an area that is blurry than one that is relatively sharper or in some cases not even visible) and that the user is less likely to spot interesting objects in its periphery (which otherwise could motivate the user to move its head towards such an object). For these reasons, at least for the *exploration* video, increased head movements can be a positive indication that the user is more engaged. Of course, for the *rides* video, this may not be the case as the expected focus typically is forward. Due to the difference in head movements, the scales of the y-axes of Figures 6(c) and 6(f) differ by a factor 20. While our observation that the *intermediate* policy always outperforms the *aggressive* policy holds in all scenarios (and across all metrics), the biggest absolute differences in head movements are observed with the *exploration* video (top row). Given the reduced head movements, it is perhaps not surprising that the *intermediate* policy performs relatively better than the *non-biased* policy for the *rides* video than for the *exploration* video.

Network conditions: Both MS-SSIM metrics considered (Figures 6(a), (b), (d), (e)) and the head movement metric for the *exploration* video (Figure 6(c)) improves with improved bandwidth conditions (poor-to-good), regardless of prefetching policy (3× cases).

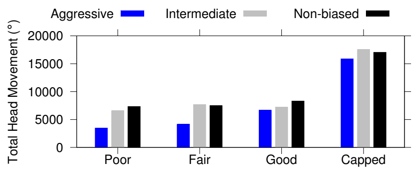
Use of bandwidth caps: Using our framework, we also provide the first initial confirmation that claims made for linear DASH video [13] suggesting that the use of bandwidth caps can reduce the



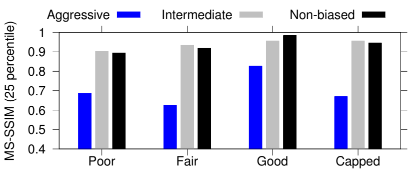
(a) Exploration, 25 percentile of MS-SSIM



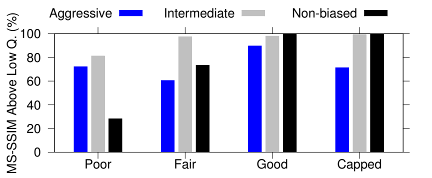
(b) Exploration, % MS-SSIM above low quality threshold



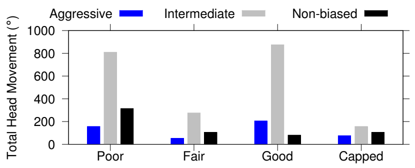
(c) Exploration, Total head movement in degrees



(d) Ride, 25 percentile of MS-SSIM



(e) Ride, % MS-SSIM above low quality threshold



(f) Ride, Total head movement in degrees

Figure 6: Scenario-based performance comparisons. (Note different y-scales in (c) and (f).)

number of stalls and quality switches also holds for 360° streaming. For example, we did not observe any stalls for any of the six *capped* scenarios, but observed at least one stall in 15 out of 18 other scenarios. On average the *poor*, *fair*, and *good* bandwidth scenarios were stalled for 3.0%, 3.0%, and 3.7% of their respective playback durations. We have also found that the smoother playback achieved by applying caps also can have positive effects on the overall video quality (e.g., MS-SSIM values for the *capped* scenario are more similar to those of the *good* scenario than the *fair* scenarios, despite having much more similar average bandwidth to the *fair* scenario) and the head movements are by far the largest for the *capped* scenario when watching the *exploration* video. The last of these new observations is interesting as it suggests that both quality switches and stalls can have negative effect on the freedom that a user feels moving their head. This again highlight the value of having a user in the loop when evaluating 360° solutions.

Use of larger buffers: We have also run experiments for the *fair* bandwidth scenario using a four-second buffer target instead of a two-second target. While a bigger buffer helped reduce the number of stalls somewhat, we still observed noticeable stalls (e.g., only reduced from 3.0% to 1.8% of the playtime) and the MS-SSIM-based QoE metrics were (slightly) worse. With exception for when using the *aggressive* policy with the *exploration* video, the two QoE metrics reported on above differed by less than 2%. However, for this case, the use of a larger buffer resulted in 15% worse 25-percentile value (for MS-SSIM) and 6% less frames with an MS-SSIM above the low-quality threshold. The reduced video quality (after removing stalls) is likely due to the example algorithms used here not accounting for the added uncertainty associated with having to make prefetching decisions earlier (which result in less accurate predictions) [2].

4 RELATED WORK

Head-movement-based analysis: Several recent works have collected head-movement datasets or characterized the 360° viewer behavior [2, 4, 8, 10, 11, 16, 19]. However, none of these works provide an emulation framework that captures the quality changes that would be experienced by a viewer of tiled video playback. As we show here, this is important to capture the true user behavior.

Bandwidth saving techniques: Much of the work on addressing limited bandwidth conditions have focused on solutions that adapt the tile quality or deliver chunks with different quality for the potential future viewing directions [1, 2, 9, 12, 14, 18, 19, 28]. Additional bandwidth savings have been achieved through improved head-movement prediction [21] using machine learning [4, 17]. Sassatelli et al. [22] propose two alternate impairments that (i) reduce the playback speed of the video and/or (ii) limits the horizontal viewing field possible.

Tile-based implementation examples: Some researchers have developed and evaluated proof-of-concept implementations of tile-based streaming [14, 18, 19, 29], including recent solutions for live streaming [6, 26] that have tighter real-time requirements. In related research, Li et al. [15] have evaluated different stitching methods used to create omnidirectional images from separate images. Ballard et al. [3] leverage hardware encoders present in modern GPUs to address the processing overhead introduced by tiling. Here, we present an evaluation framework that incorporates efficient GPU-based viewport emulation.

5 CONCLUSION

This paper presents REEFT360, which consists of (1) a real-time emulation framework for 360° video that captures tile quality adaptation under time-varying bandwidth conditions and (2) a multi-step evaluation process that allows the calculation of per-frame-based quality metrics (e.g., MS-SSIM) while accounting for the user's head-movements during playback. The modular design allows speedy implementation and testing of alternative prediction and prefetching solutions, allows testing under a wide range of network conditions, and can be used both with a human user and with head-movement traces. Our proof-of-concept evaluation results highlight the importance of including a human subject in the evaluation. For example, as we demonstrate here, the head-movements in exploration scenarios can be greatly restrained by the quality degradation of some of the tiles being presented to the user. This further highlights the value of the tool presented and shared with this paper, which should help others to easily implement and evaluate their own tile-based 360° solutions with a *human in the loop.*

ACKNOWLEDGMENTS

This work was part funded by the Swedish Research Council (VR).

REFERENCES

[1] Patrice Rondao Alface, Jean-François Macq, and Nico Verzijp. 2012. Interactive omnidirectional video delivery: A bandwidth-effective approach. *Bell Labs Tech. Journal* 16, 4 (2012).

[2] Mathias Almquist, Viktor Almquist, Vengatanathan Krishnamoorthi, Niklas Carlsson, and Derek Eager. 2018. The Prefetch Aggressiveness Tradeoff in 360° Video Streaming. In *Proc. ACM MMSys*.

[3] Trevor Ballard, Carsten Griwodz, Ralf Steinmetz, and Amr Rizk. 2019. RATS: Adaptive 360-Degree Live Streaming. In *Proc. ACM MMSys*.

[4] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. 2016. Shooting a moving target: Motion-prediction-based transmission for 360 videos. In *Proc. IEEE Big Data*.

[5] N. Carlsson, D. Eager, V. Krishnamoorthi, and T. Polishchuk. 2017. Optimized Adaptive Streaming of Multi-video Stream Bundles. *IEEE Trans. on Multimedia* (2017), 1637–1653.

[6] Bo Chen, Zhisheng Yan, Haiming Jin, and Klara Nahrstedt. 2019. Event-driven stitching for tile-based live 360 video streaming. In *Proc. ACM MMSys*.

[7] Shyamprasad Chikkerur, Vijay Sundaram, Martin Reisslein, and Lina J Karam. 2011. Objective video quality assessment methods: A classification, review, and performance comparison. *IEEE Trans. on Broadcasting* 57, 2 (2011), 165–182.

[8] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-Degree Video Head Movement Dataset. In *Proc. ACM MMSys*.

[9] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *Proc. IEEE ICC*.

[10] Erwan J. David, Jesús Gutiérrez, Antoine Coutrot, Matthieu Perreira Da Silva, and Patrick Le Callet. 2018. A Dataset of Head and Eye Movements for 360°; Videos. In *Proc. ACM MMSys*.

[11] Stephan Fremerey, Ashutosh Singla, Kay Meseberg, and Alexander Raake. 2018. AVtrack360: An Open Dataset and Software Recording People's Head Rotations Watching 360°; Videos on an HMD. In *Proc. ACM MMSys*.

[12] M. Hosseini and V. Swaminathan. 2016. Adaptive 360 VR video streaming: Divide and conquer. In *Proc. IEEE ISM*.

[13] Vengatanathan Krishnamoorthi, Niklas Carlsson, and Emir Halepovic. 2018. Slow but Steady: Cap-based Client-Network Interaction for Improved Streaming Experience. In *Proc. IEEE/ACM IWQoS*.

[14] Jean Le Feuvre and Cyril Concolato. 2016. Tiled-Based Adaptive Streaming Using MPEG-DASH. In *Proc. ACM MMSys*.

[15] Jia Li, Kaiwen Yu, Yifan Zhao, Yu Zhang, and Long Xu. 2019. Cross-Reference Stitching Quality Assessment for 360° Omnidirectional Images. In *Proc. ACM Multimedia*.

[16] W. Lo, C. Fan, J. Lee, C. Huang, K. Chen, and C. Hsu. 2017. 360° Video Viewing Dataset in Head-Mounted Virtual Reality. In *Proc. ACM MMSys*.

[17] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. 2018. Your Attention is Unique: Detecting 360-Degree Video Saliency in Head-Mounted Display for Head Movement Prediction. In *Proc. ACM Multimedia*.

[18] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos. In *Proc. ACM Multimedia*.

[19] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proc. ACM MobiCom*.

[20] Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. 2018. Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics. In *Proc. ACM MMSys*.

[21] Miguel Fabián Romero Rondón, Lucile Sassatelli, Ramon Aparicio-Pardo, and Frédéric Precioso. 2020. A unified evaluation framework for head motion prediction methods in 360° videos. In *Proc. ACM MMSys*.

[22] L. Sassatelli, M. Winckler, T. Fisichella, A. Dezarnaud, J. Lemaire, R. Aparicio-Pardo, and D. Trevisan. 2020. New interactive strategies for virtual reality streaming in degraded context of use. *Computers and Graphics* (2020), 27–41.

[23] Ashutosh Singla, Steve Göring, Alexander Raake, Britta Meixner, Rob Koenen, and Thomas Buchholz. 2019. Subjective quality evaluation of tile-based streaming for omnidirectional videos. In *Proc. MMSys*.

[24] Jangwoo Son, Dongmin Jang, and Eun-Seok Ryu. 2018. Implementing Motion-Constrained Tile and Viewport Extraction for VR Streaming. In *Proc. ACM NOSSDAV*.

[25] Thomas Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP: Standards and Design Principles. In *Proc. ACM MMSys*.

[26] Liyang Sun, Yixiang Mao, Tongyu Zong, Yong Liu, and Yao Wang. 2020. Flocking-based live streaming of 360-degree video. In *Proc. ACM MMSys*.

[27] Z. Wang, E. P. Simoncelli, and A. C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Asilomar Conference on Signals*. 1398–1402.

[28] Praveen Kumar Yadav and Wei Tsang Ooi. 2020. Tile Rate Allocation for 360-Degree Tiled Adaptive Video Streaming. In *Proc. ACM Multimedia*.

[29] Alireza Zare, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. 2016. HEVC-Compliant Tile-Based Streaming of Panoramic Video for Virtual Reality Applications. In *Proc. ACM Multimedia*.