Robust 360° Video Streaming via Non-Linear Sampling

Mijanur Palash, Voicu Popescu, Amit Sheoran, Sonia Fahmy
Purdue University
{mpalash, popescu, asheoran, fahmy}@purdue.edu

***Abstract*—We propose CoRE, a 360° video streaming approach that reduces bandwidth requirements compared to transferring the entire 360° video. CoRE uses non-linear sampling in both the spatial and temporal domains to achieve robustness to view direction prediction error and to transient wireless network bandwidth fluctuation. Each CoRE frame samples the environment in all directions, with full resolution over the predicted field of view and gradually decreasing resolution at the periphery, so that missing pixels are avoided, irrespective of the view prediction error magnitude. A CoRE video chunk has a main part at full frame rate, and an extension part at a gradually decreasing frame rate, which avoids stalls while waiting for a delayed transfer. We evaluate a prototype implementation of CoRE through trace-based experiments and a user study, and find that, compared to tiling with low-resolution padding, CoRE reduces data transfer amounts, stalls, and H.264 decoding overhead, increases frame rates, and eliminates missing pixels.**

I. Introduction

360° videos allow users to vividly experience complex environments. The user can actively change view direction, which affords a greater sense of presence in the environment compared to passively watching a conventional video that chooses the view direction for the user.

Challenges. A 360° video has to store a large number of pixels to capture the environment in all directions. To reduce bandwidth requirements over wireless links, only the part of the 360° video that the user actually sees is transferred. For example, for a user Field of View (FoV) of 90° × 90°, the user only sees about one eighth of a 360° × 180° equirectangular frame. In this case, 360° video streaming has to predict the user view direction [17].

A second challenge for wireless 360° video streaming is the fluctuation of network bandwidth, caused by handovers between radio towers, or by sparse radio coverage [1], [23]. For example, a mobile device on a high speed train can experience connectivity failures of 2 to 15s, every 150s [22]. As cellular carriers strive for higher bandwidth, the initially sparse coverage of 5G radio [19] will exacerbate bandwidth fluctuation and make bandwidth prediction difficult. Bandwidth fluctuation can be masked in conventional video streaming by prefetching and rate adaptation [2], [27]. However, prefetching in 360° video streaming is constrained by how far ahead one can reliably predict the user view direction. Delayed prefetching causes playback *stalls* and loss of viewership [10].

This work has been supported in part by NSF grant CNS-1717493.

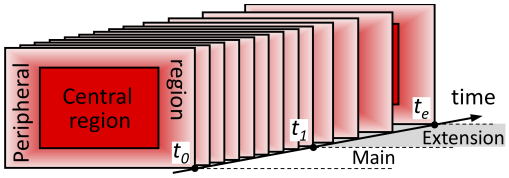


Fig. 1: CoRE 360° video concept.

State of the Art. Most 360° video streaming systems [8], [17] partition the input video uniformly into rectangular tiles, and transfer the tiles that overlap with the predicted user FoV, plus a safety margin, possibly at a lower resolution. For example, in a 4 × 6 tiling of an equirectangular video, a tile covers about 60°×45°, so a 90°×90° FoV requires at least 2×2 tiles, which corresponds to a transfer size reduction of at most $24/4 = 6$. A safety margin of only 10° brings the FoV to 110°×110°, for a transfer size reduction of $24/(3 \times 3) = 2.66$. Using lower resolution tiles at the periphery of the FoV reduces transfer size, but introduces abrupt resolution changes.

To increase view prediction accuracy, 360° video streaming systems reduce the duration of the video *chunk* from 4s [9] to 1s [8], [17]. This implies delayed prefetching and more likely stalls. Further, a shorter chunk yields more files, e.g., 36 files for 4s of 3 × 3 1s tiles. This decreases compression effectiveness as the codec cannot exploit data redundancy across spatial and temporal tile borders, and increases decompression cost at the client. This can be alleviated with variable-sized disjoint [6] or overlapping [29] tiles, which allow partitioning with uniform complexity [6], and allow managing storage requirements at the server [29], while capping the overall number of tiles, albeit to about 30 tiles. Most state-of-the-art systems [6], [8], [17] require a GPU, which drains batteries and precludes thin clients.

Proposed Approach. We propose CoRE (**Co**mpressed **R**otated **E**quirectangular) 360° video streaming, designed to satisfy five main requirements:

(1) *Robustness to view prediction error:* no missing pixels, regardless of view prediction error size.

(2) *Robustness to transient bandwidth fluctuation:* minimize stalls when the next chunk is delayed.

(3) *Single file:* reconstruct each frame from one video file.

(4) *Compatibility with current streaming:* leverage standard codecs and protocols, e.g., H.264 and DASH [9].

(5) *Computational efficiency:* have low computational cost to support clients with no GPU.

CoRE is based on a 360° video parameterization designed for streaming. *Spatially*, a CoRE frame covers all view directions, so there are no missing pixels, regardless of view prediction error magnitude. A CoRE frame has a central region at full resolution, and a peripheral region at gradually decreasing resolution (Figure 1). The central region is aligned with the user view direction, so output frame pixels inside the predicted FoV are computed from full resolution data. Pixels outside the predicted FoV are computed from the lower resolution peripheral region. The peripheral region's non-linear resolution provides higher quality for smaller prediction errors, which are more frequent. The CoRE frame resolution continuity avoids abrupt quality changes.

Temporally, a CoRE chunk has a *main* part at full frame rate (from t_0 to t_1 in Figure 1), followed by an *extension* part at a gradually decreasing frame rate (from t_1 to t_e). In the absence of bandwidth fluctuation, the output video is reconstructed using the main part, at full frame rate. If the next chunk is delayed, stalls are avoided for up to $t_e - t_1$ seconds by reconstructing the output video from the extension part. The extension's non-linear frame rate handles the more likely short delays without significant frame rate decreases. When the extension is used, the scene captured by the video is updated less frequently, but the system remains fully responsive to user view direction changes, without the visual degradation implied by conventional bit rate reduction. The user can rotate the view direction with no latency, with the only difference being that dynamic scene objects are updated 10 and not 30 times a second, for example.

The CoRE non-linear spatial and temporal sampling buys insurance against degradation caused by view prediction error and by stalls. The insurance premium is the transfer of additional data in the periphery region and the extension part. The sampling scheme provides the flexibility needed to minimize the additional transfer and maximize output quality, based on the application, the scene, and the network conditions.

CoRE chunks are compressed with a standard low-level video codec, e.g., H.264, and transferred to the client with a standard video streaming protocol, e.g., DASH. The server precomputes chunks for a uniform discretization of all possible view directions, which trades off storage for scalability with the number of clients, conforming to the DASH paradigm. A single CoRE frame covers the entire FoV, which maximizes video compression efficiency, and minimizes client video decoding costs. Finally, CoRE optimizes the mapping from the output frame to the CoRE frame, achieving 30fps on a tablet in a purely serial CPU implementation, whereas most prior systems can only work with a GPU. This opens the door to 360° video streaming on the thinnest of clients.

We have evaluated CoRE on several 360° videos, user view direction traces, and wireless network traces, compared it to five baseline tiling variants, and conducted a user study. Our results indicate that CoRE significantly reduced the amount of data transferred, stalls, and H.264 decoding overhead, eliminated missing pixels and abrupt resolution changes, and increased frame rates, compared to tiling with padding.

II. CoRE Streaming

Figure 2 shows the architecture of the CoRE 360° video streaming system. The 360° video content is encoded offline (1-2) into CoRE chunks (§II-A) for a discretization of all possible view directions. Then, while streaming online, at a time determined adaptively (§II-E), the client requests a CoRE chunk from the server (6-9) based on the predicted view (5), which is based on the current user view (4), captured by the *View User Interface* (3). The client decodes the CoRE chunk (A, §II-C) using the current user view (B) to compute the current output frame (C).

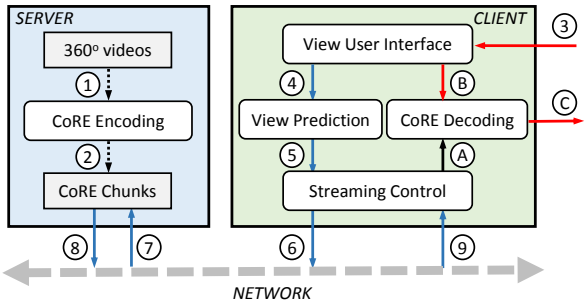


Fig. 2: CoRE architecture.

A. CoRE Encoding

A CoRE chunk is a video cube that covers a central region at full resolution and peripheral regions at decreasing spatial and temporal resolutions (Figure 1). A CoRE chunk is constructed with algorithm 1. The input video V^* is encoded with a standard video codec, such as H.264. Any omnidirectional parameterization is supported, such as equirectangular, cube map, fisheye, or other spherical projections; all that is needed is a function that projects a direction onto the input 360° video frame, as discussed below. $w \times h$ is chosen to match the output client frame resolution, but not to exceed the resolution available in the input video V^*. w_e, Δt_e, and n_e, balance the robustness/overhead trade-off.

Algorithm 1: CoRE Chunk Encoding

Input: input 360° video V^*, predicted user view direction d_p, user FoV $\Delta\phi \times \Delta\theta$, CoRE chunk starting time t_0, main duration Δt, central region resolution $w \times h$, peripheral region lateral thickness w_e, extension duration Δt_e, and number of extension frames n_e.
Output: CoRE video chunk V^*_{CoRE}.

1 (V, ω) = *LowLevelDecode*$(V^*, t_0, t_0 + \Delta t + \Delta t_e)$
2 **for** i = *0 to* $\omega\Delta t$ **do**
3 $\quad V_{CoRE}[i]$ = *CoREFrameEncd*$(V[i], d_p, \Delta\phi, \Delta\theta, w, h, w_e)$
4 (a_0, a_1, a_2) = *SetupFrameRateDecrease*$(\omega, \Delta t_e, n_e)$
5 **for** j = *1 to* n_e **do**
6 $\quad t_j = a_0 j^2 + a_1 j + a_2; k = \lfloor \omega(\Delta t + t_j) \rfloor$
7 $\quad V_{CoRE}[i{+}{+}]$ = *CoREFrameEncd*$(V[k], d_p, \Delta\phi, \Delta\theta, w, h, w_e)$
8 **return** V^*_{CoRE} = *LowLevelEncode*(V_{CoRE})

The algorithm extracts (from the input video) the frames within the main and extension time intervals (line 1). The

algorithm computes the $\omega\Delta t$ frames of the main part, where ω is the frame rate of the input video (lines 2-3). Each CoRE frame is computed from one input video frame, as described in §II-B. The main part has a constant frame rate, ω.

The extension part frames are placed on the timeline farther and farther apart (Figure 1). Let the timeline position of frame j of the extension be $t(j)$, with respect to the beginning of of the extension. Then $t(j)$ has to satisfy three conditions (Equation 1): the first frame has to be placed at the beginning of the extension, the last frame has to be placed at the end of the extension, and the frame rate at the beginning of the extension part has to match that of the main part, i.e., ω.

$$t(0) = 0, t(n_e) = \Delta t_e, t'(0) = 1/\omega \tag{1}$$

Satisfying three conditions requires three coefficients, so the simplest expression for $t(j)$ is a quadratic (Equation 2).

$$t(j) = a_0 j^2 + a_1 j + a_2, t'(j) = 2a_0 j + a_1 \tag{2}$$

The coefficients a_i are found (Equation 3, line 4) by solving the system of linear equations defined by Equations 1 and 2. The lowest frame rate over the extension part of the CoRE chunk occurs at the end of the extension, and it is equal to $1/t'(n_e)$. The average frame rate over the extension is $n_e/\Delta t_e$.

$$a_0 = (\omega\Delta t_e - n_e)/(\omega n_e^2), a_1 = 1/\omega, a_2 = 0 \tag{3}$$

The algorithm computes each extension frame j from its corresponding input video frame k (lines 5-7). k is found by computing the extension timeline position t_j of frame j and then by finding the input frame at time point $\Delta t + t_j$ (line 6). Finally, the CoRE chunk is encoded with a codec (e.g., H.264) to reduce file size by data compression (line 8).

B. CoRE Frame Encoding

A CoRE frame is computed as follows (Figure 3 a-d). The input equirectangular frame (a) is rotated (b) to center the predicted user FoV (black). Then, the peripheral region is compressed. The resulting CoRE frame is shown at scale in c, and magnified in d, with the central region shown in red.

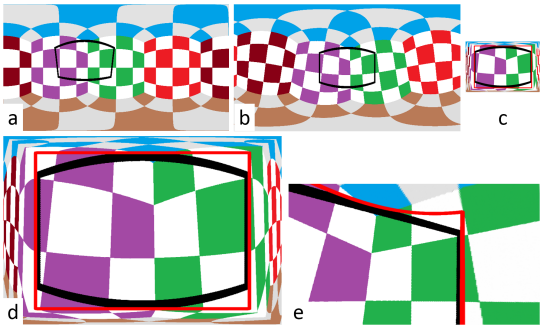


Fig. 3: CoRE frame construction stages (a-d), and output frame (e) decoded from CoRE frame (d).

A CoRE frame is computed from an input 360° frame F according to algorithm 2. The vertical thickness of the peripheral region h_e is computed using the number of pixels per degree defined by the input parameter w_e (line 1). For

Algorithm 2: CoRE Frame Encoding

Input: 360° video frame F, predicted user view direction d_p, user FoV $\Delta\phi \times \Delta\theta$, central region resolution $w \times h$, and peripheral region lateral thickness w_e.
Output: CoRE frame F_{CoRE}

1 $h_e = w_e(180^o - \Delta\theta)/(360^o - \Delta\phi)$
2 $u_0 = w(360^o/\Delta\phi - 1)/2; v_0 = h(180^o/\Delta\theta - 1)/2$
3 $R.x = d_p; R.y = d_p \times (0,1,0); R.z = R.x \times R.y$
4 **for** $v = 0$ *to* $h + 2h_e$ **do**
5 **for** $u = 0$ *to* $w + 2w_e$ **do**
6 **if** $(u,v) \in [w_e, w_e + w] \times [h_e, h_e + h]$ **then**
7 $(u_e, v_e) = (u,v) - (w_e, h_e) + (u_0, v_0)$
8 **else**
9 (u_e, v_e) = Expand(u,v)
10 d_e = Unproject(u_e, v_e)
11 $d_r = R^T d_e^T$
12 (u_i, v_i) = Project(d_r)
13 $F_{CoRE}(u,v)$ = LookUp(F, u_i, v_i)
14 **return** F_{CoRE}

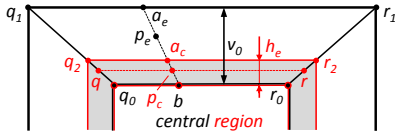


Fig. 4: Expansion of CoRE peripheral region.

example, for a 90° × 90° user FoV, the vertical thickness of the peripheral region is $w_e/3$. The coordinates (u_0, v_0) of the top left corner of the central region in the expanded equirectangular frame are computed as half of the total width and height of the expanded frame (line 2). In our 90° × 90° FoV example, for a w of 1,000, the total width of the expanded frame is $1,000(360°/90°) = 4,000$, and u_0 is 1,500. The local coordinate system of the user FoV is constructed from the predicted view direction (line 3).

The CoRE frame is computed one pixel at the time (lines 4-13). If the current pixel (u,v) is inside the core region, its coordinates in the expanded equirectangular frame are computed by adding an offset that takes into account the different peripheral region thickness (line 7). If the current pixel is in the peripheral region (line 9), its expanded coordinates are computed as described below.

The direction d_e to the center of the current pixel (u,v) is computed by unprojecting the equirectangular frame 2D point (u_e, v_e) to a 3D point d_e on the unit sphere (line 10). d_e is rotated back to the world coordinate system to obtain d_r (line 11). Finally, d_r is projected to the input 360° frame F (line 12), where color is looked up (line 13). For example, if F is equirectangular, the projection computes the latitude/longitude values of a point given on the unit sphere.

Figure 4 illustrates the expansion of a point p_c inside the top part of the compressed peripheral region (shaded) to expanded equirectangular frame point p_e. The expansion proceeds as follows. Find points p and q at the intersection of segments q_1q_0 and r_1r_0 with the horizontal line through p_c. Find point a_c such that $q_2a_c/a_cr_2 = qp_c/p_cr$. Find points a_e and b by intersecting a_cp_c with q_1r_1 and q_0r_0, respectively. The

expanded location p_e of p_c belongs to a_eb. p_e is placed on a_eb by computing the vertical coordinate v_e of p_e. v_e is a function $v_e(v_c)$, which has to satisfy three conditions (Equation 4).

$$v_e(h_e) = v_0, v_e(0) = 0, v'_e(h_e) = 1 \quad (4)$$

The first two conditions define the expansion for the inner and outer edges of the peripheral region (e.g., the coordinate of b changes from h_e to v_0, and the coordinate of a_c remains 0 when expanded to a_e). The third condition ensures that the CoRE sampling rate is continuous from the central to the peripheral region. A quadratic accommodates all three conditions (Equation 5).

$$v_e(v_c) = a_0v_c^2 + a_1v_c + a_2, v'_e(v_c) = 2a_0v_c + a_1 \quad (5)$$

The coefficients, computed using Equation 4, are shown in Equation 6. Along segment ba_c, the expansion starts slowly, with no scaling close to b, and then picks up, to push a_c all the way to a_e. The worst sampling rate is at the outer edge of the CoRE frame, where a one pixel step corresponds to a $2v_0/h_e - 1$ pixel step in the expanded frame. The other three parts of the peripheral region are expanded similarly.

$$a_0 = (h_e - v_0)/h_e^2, a_1 = 2v_0/h_e - 1, a_2 = 0 \quad (6)$$

C. CoRE Frame Decoding

The client computes output frames from the CoRE chunk. The chunk is first low-level decoded to extract the CoRE frames. Then each output frame is computed from its corresponding CoRE frame. The main part of the CoRE chunk is played back. Then, if the next chunk is delayed, the extension part is played back, using Equations 2 and 3 to find the index j of the CoRE frame to be used for a given output frame.

An output frame F_{out} is computed by decoding the CoRE frame F_{CoRE} with algorithm 3. Matrix C_f denotes the current user view. Each pixel (u, v) of F_{out} is computed by finding its corresponding pixel (u_c, v_c) in F_{CoRE} (lines 1-7). The mapping unprojects the current pixel (u, v) to a 3D direction d (line 3), d is rotated to d_r in the CoRE coordinate system (line 4), d_r is projected onto the equirectangular frame at (u_e, v_e) (line 5), and (u_e, v_e) is moved to CoRE coordinates (u_c, v_c) by compressing the peripheral region (line 6). The compression is the inverse of the expansion described in §II-B. Pixel (u, v) is set by bilinear interpolation of F_{CoRE} at (u_c, v_c) (line 7).

Figure 3e shows an output frame decoded from the CoRE frame (d). Due to view prediction error, the actual FoV extends beyond the predicted view (black line in e) and the CoRE central region (red line in e). The output frame (e) has full resolution where core region pixels are used (left and below red line), and smoothly decreasing resolution where peripheral region pixels are used (right and above red line).

D. CoRE Frame Decoding Optimization

We optimize the mapping from the output frame to the CoRE frame as follows. Referring back to algorithm 3, the camera matrix C_f is pre-rotated by multiplication with R ($M = RC_f$). Then d_r is computed incrementally at the cost of three floating point adds, i.e. $d_r(u, v) = d_r(u - 1, v) + M_{i2}$, where M_{i2} is the third column of M. The equirectangular projection (line 5) normalizes d_r, finds the latitude v_e, which depends only on $d_r.y$, and finds the longitude u_e, which depends only on $d_r.x$ and $d_r.z$. The latitude and longitude are found in pre-computed 1D and 2D look up tables, respectively. The compression (line 6) is also looked up in a pre-computed 2D table. Consequently, the cost of computing an output frame pixel is 3 adds, one 3D vector normalization, 3 table look-ups, and a bilinear interpolation.

Algorithm 3: CoRE Frame Decoding

Input: CoRE video frame F_{CoRE}, central region resolution $w \times h$, peripheral region lateral and vertical thickness w_e and h_e, rotation matrix R from world to CoRE local coordinate system, current user view camera matrix C_f, and output frame resolution $w_f \times h_f$.
Output: frame F_{out} displayed by client for user.

```
1 for v = 0 to h_f do
2     for u = 0 to w_f do
3         d = C_f(u, v, 1)^T
4         d_r = Rd
5         (u_e, v_e) = Project(d_r)
6         (u_c, v_c) = Compress(u_e, v_e)
7         F_out(u, v) = LookUp(F_CoRE, u_c, v_c)
8 return F_out
```

E. Adaptive Prefetching

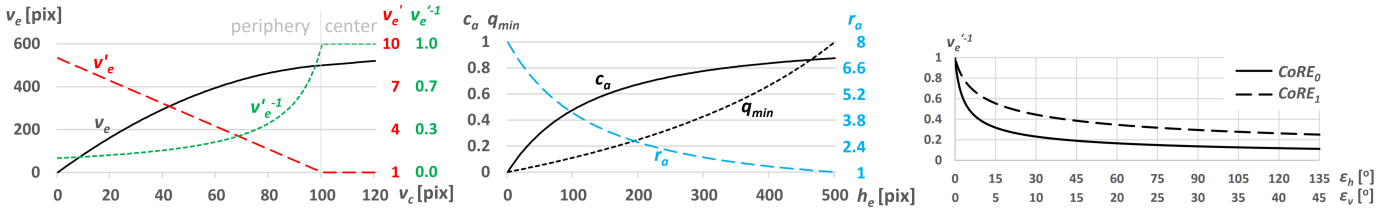
We adaptively determine the time to prefetch each CoRE file, inspired by the TCP Retransmission Timeout (RTO) computation algorithm [14]. We update the prefetching time, which we subtract from the end time of the current chunk, as: Smoothed download time = $(1 - \alpha)$ × Smoothed download time $+\alpha\times$ Last download time; Download time variation = $(1-\beta)$ × Download time variation $+\beta\times$ (Smoothed download time − Last download time); and Prefetching time = $\gamma\times$ Smoothed download time $+\rho\times$ Download time variation.

We experimented with different values of α, β, γ, and ρ. We varied α and β between 0 and 1 in increments of 0.1, and γ and ρ from 0.5 to 2 in increments of 0.5. We also compared with using fixed prefetching times of 1s, 2s, 3s, or 4s before the end of the current chunk. We found that adaptive prefetching with $\alpha = 0.9$, $\beta = 0.9$, $\gamma = 0.5$ and $\rho = 1$ worked best. We therefore use these values in all our experiments in this paper.

III. CoRE Streaming Analysis

In this section, we analyze the CoRE cost/benefit trade-offs.

Robustness to view prediction error. A CoRE frame is a 360° frame, so there are no missing pixels, regardless of the magnitude of the view prediction error. However, the larger the view prediction error, the farther from the central region the CoRE frame is sampled, and the lower the image quality. Given a pixel (u, v) in the output frame that is looked up in the CoRE frame at (u_c, v_c), we quantify image quality at (u, v) as the sampling rate at (u_c, v_c). The sampling rate $q(v_c)$ is the inverse of the sampling step along the compression direction, i.e., $q(v_c) = v'_e(v_c)^{-1} = 1/v'_e(v_c)$.



(a) (b) (c)

Fig. 5: CoRE sampling rate vs. periphery coordinate (a) and thickness (b), and vs. view prediction error (c).

Assume the central region has an FoV of $\Delta\phi \times \Delta\theta = 90° \times 90°$, and a resolution of $w \times h = 1,000 \times 1,000$. If the lateral thickness of the peripheral region is $w_e = 300$, based on lines 1-2 of algorithm 2, the vertical thickness is $h_e = 100$, and the thicknesses of the expanded peripheral region are $u_0 = 1,500$ and $v_0 = 500$. Figure 5a shows the dependence of the expanded vertical coordinate v_e, of the sampling step v'_e, and of the sampling rate v'^{-1}_e, on the CoRE vertical coordinate v_c, over the peripheral region (e.g., from a_e to b in Figure 4). v_e has a large initial velocity and decelerates uniformly until the boundary between the peripheral and central regions, after which it changes one-to-one with v_c. The sampling step v'_e starts out at $v'_e(0) = 9$ and decreases linearly to become $v'_e(100) = 1$ at the boundary. The sampling rate v'^{-1}_e starts out as $1/9$ and increases to 1 at the boundary. The average sampling step (rate) is $v_0/u_0 = 5$ $(1/5 = 0.2)$.

The cost/benefit trade-off is controlled through the thickness of the peripheral region w_e, which also determines h_e. Figure 5b shows the minimum quality q_{min}, defined as the sampling rate $v'_e(0)^{-1}$ at the border of the CoRE frame, the CoRE overhead c_a, defined as the area of the peripheral region over the area of the entire CoRE frame, and the CoRE size reduction r_a, defined as the area of the expanded frame over the area of the CoRE frame. q_{min} corresponds to the largest possible view prediction error, i.e., when the user looks in the direction opposite to the one predicted. For a thin border with $h_e = 100$pix, q_{min} is 1/9, the peripheral region is 48% of the CoRE frame ($c_a = 0.48$), and the CoRE frame is over 4 times smaller than a conventional 360° frame ($r_a = 4.16$). A border of $h_e = 200$pix raises q_{min} to 0.25, at the cost $c_a = 0.67$, while still achieving a reduction $r_a = 2.6$.

We now analyze the dependence of the sampling rate on the magnitude of view prediction error. For our example, the 270° of horizontal FoV outside of the central region are distributed evenly left and right, hence the largest possible horizontal view prediction error (ϵ_h) is 135°. Similarly, the largest possible vertical view prediction error (ϵ_v) is 45°. In Figure 5c, the $CoRE_0$ configuration has $h_e = 100$, $c_a = 0.48$, and $r_a = 4.16$. The sampling rate is above 0.25 for horizontal (vertical) view direction prediction errors of up to 15° (5°), and above 0.2 for errors up to 45° (15°). The $CoRE_1$ configuration has $h_e = 200$, $c_a = 0.67$, and $r_a = 2.6$, and the sampling rate is above 0.4 for errors up to 45° (15°).

Robustness to link bandwidth fluctuation. The cost/benefit trade-off is controlled through two parameters: the duration Δt_e and the number of frames n_e in the CoRE extension. Δt_e should be selected to avoid stalls for the longest anticipated bandwidth degradation, e.g., handovers between cellular towers. A CoRE chunk can avoid stalls for up to $\Delta t + \Delta t_e$, where Δt is the duration of the main. n_e should be selected based on the overhead the system or application deems acceptable, and based on the desired average ω_{avg} and minimum ω_{min} frame rates over the extension. We define the overhead c_e as the ratio between n_e and the number of main part frames $\omega\Delta t$, where ω is the main part frame rate.

ω_{avg} is linear in n_e (i.e., $\omega_{avg} = n_e/\Delta t_e$), as is c_e (i.e., $c_e = n_e/(\omega\Delta t)$). ω_{min} is one over linear in n_e ($\omega_{min} = t'(n_e)^{-1}$, where t is defined by Equation 2 and Equation 3). For 30 extra frames and a 6s extension, the frame rate decreases from 30fps to 2.72fps, for an average frame rate of 5fps, and an overhead of 25%. For a CoRE configuration with 60 extra frames, the overhead increases to 50%, and the average and minimum frame rates increase to 10fps and 6fps, respectively.

IV. Experimental Evaluation

We first give our evaluation methodology, then our results.

A. Evaluation Methodology

We conduct trace-based experiments comparing C++ client and server implementations of CoRE and tiling, running on the CPU of an HP EliteBook 8470p laptop with Intel core i5 3320M 2.6 GHz dual core processor and 8 GB of RAM. Our C++ implementations consist of 5,000+ LoC. We also conduct a user study using an Android client prototype on a Samsung galaxy Tab S6 with Qualcomm snapdragon 855 Octa-core (1x2.84GHz + 3x2.41GHz + 4x1.78GHz) processor and Android version 9. Our Android implementation reuses our C++ code with JNI, with an additional 1,000+ Java LoC.

1) Streaming Approaches: We compare CoRE streaming with the following tiling baseline algorithms for streaming a stored video: (1) FoV only (fetch only the tiles overlapping with the user field of view), (2) FoV+ 1QL (also fetch padding tiles outside field of view, with all tiles of the same quality level), (3) FoV+ 2QL (padding tiles have a lower quality level than FoV tiles), (4) FoV 360 (fetch all tiles, with higher quality FoV tiles), and (5) FoV+ 360 (fetch all tiles, with higher quality FoV and padding tiles).

In our tiling implementations, we do not skip any of the video frames when stalls occur; rather, we play back the entire

video, which may require a longer time to play back than the video duration, due to stalls. We also stop head movement. Hence, the results we report in terms of tiling frame quality are advantageous to tiling. We also incorporate the decoding optimizations that we developed for CoRE (§II-D) into the tiling implementations for a fair comparison.

FoV+ 2QL, FoV 360, and FoV+ 360 follow the multi-resolution approach taken by state-of-the-art 360° video streaming systems [6], [8], [17]. Since we are unable to obtain implementations of these systems, our comparison remains at the *algorithmic* level, without the optimizations that these systems employ. We note that CoRE is *compatible* with several of these optimizations, as discussed in §VI. CoRE merely mitigates the challenging problems of view direction prediction and transient bandwidth fluctuation.

For tiling, unless otherwise specified, we use the following parameters that are used in Flare [17]: (1) duration of each video chunk is 1s, (2) user FoV is $90^\circ \times 48^\circ$, (3) number of tiles the video is partitioned into is 4×6, (4) amount of padding is 20%, and (5) All high quality tiles have sampling rate 1, and low quality tiles have sampling rate 0.5.

For CoRE, unless otherwise specified, we use the following parameters: (1) user output frame resolution $w_f \times h_f$ is 960×512, (2) user FoV $\Delta\phi \times \Delta\theta$ is $90^\circ \times 48^\circ$, (3) resolution of the encoded high quality central region of CoRE $w \times h$ is $1,152 \times 614$, (4) resolution of the full CoRE frame with high-quality center and peripheral region is $1,688 \times 1,068$, (5) maximum sampling rate is 1 (central area) and minimum sampling rate is 0.11 in a CoRE frame, (6) we request the next CoRE adaptively within 1s-4s before the playback of the main part of the current CoRE chunk will be completed, (7) the real time duration of each CoRE is 10s: a 4s main part plus a 6s extension part, and (8) the frame rate ω of the main part is 30fps, and the extension part has $n_e = 30$ frames.

2) Videos: We use six videos obtained from YouTube for our experiments: (1) underwater diving (Diving), (2) elephants drinking from a lake (Elephant), (3) panoramic view of New York City (NY), (4) panoramic view of Paris (Paris), (5) rhinoceroses (Rhino), and (6) a roller coaster ride (Roller). All videos have approximately 4k (3,840×2,048) resolution. We take approximately one minute of each video (but we use two minutes for the user study), convert to .avi format, and re-encode with Constant Rate Factor (CRF) 30. The resulting file sizes are listed in Table I.

TABLE I: Videos used in experiments

Video	YouTubeID	File Size (MB)		
		Original	4s chunks	1s chunks
Diving	2OzlksZBTiA	69.5	69.6	75.15
Elephant	2bpICIClAIg	39.6	40.0	43.8
NY	CIw8R8thnm8	46.7	48.5	55.3
Paris	sJxiPiAaB4k	15.2	18.5	36.2
Rhino	7IWp875pCxQ	13.2	15.2	26.75
Roller	8lsB-P8nGSM	49.3	49.7	55.22

3) Head Movement: We use one-minute head movement traces [4] based on the work from Corbillon *et al.* [3]. We list the total number of traces for each video in Table II. Except for §IV-B6, we use the *current view direction* information when fetching. This represents no head movement prediction, and is a challenging case for both tiling and CoRE.

TABLE II: Head movement traces used in experiments

Video	Diving	Elephant	NY	Paris	Rhino	Roller
Traces	60	38	57	58	25	59

4) Network Conditions: For our trace-based experiments, we use bandwidth traces obtained from the Mahimahi project repository [12], which were collected using the Saturator tool by Winstein *et al.* [23]. We choose these traces because, to the best of our knowledge, they have the highest precision, i.e., millisecond level, among all publicly available traces. The four traces we use are listed in Table III. Note that the average bandwidth listed in the table is in KBps, for ease of comparison with transfer amounts that are given in Bytes. The average is computed over approximately the first 60 seconds, since this is the length of the head movement traces [4].

TABLE III: Bandwidth traces used in experiments

Trace ID	Avg Bandwidth (KBps)
TMobile-UMTS-driving.down	121
ATT-LTE-driving-2016.down	576
TMobile-LTE-driving.down	1593
Verizon-LTE-driving.down	1899

5) Performance Metrics: We compare CoRE to tiling in terms of the quality/cost trade-off. Quality is measured via three intra-frame (a-c) and two inter-frame (d-e) metrics, and cost is measured via two metrics (f-g).

(a) **Missing pixels**: an output frame pixel that maps outside the fraction of the 360° frame transferred from the server.

(b) **Sampling rate**: inverse of sampling step (§III), ×100, over all pixels of all output frames.

(c) **Abrupt changes in sampling rate**: number of output image pixels that have at least one neighboring pixel of a lower quality level, defined for approaches that use more than one discrete resolution level.

(d) **Frame rate**: total number of frames played divided by the playback duration. For tiling baselines, we consider the next frame ready if *at least* the FoV tiles are ready, even if padding tiles are not yet ready.

(e) **Stalls**: a stall occurs when the output frame cannot be updated because the playback time has reached the last time step covered by the data received. For CoRE, a stall starts when the last frame of the extension is used. For tiling, a stall starts when the last frame of the current chunk is used; again, we do not wait for the arrival of the padding tiles.

(f) **Amount transferred:** total amount of data transferred over the entire experiment duration.

(g) **Low-level decoding overhead:** time and energy consumed by the video codec for low-level decoding at the client.

Unlike our detailed quality metrics (a-c), a lump-sum metric such as PSNR would not detect and quantify errors that occur over a small number of pixels, such as sampling rate discontinuities, or a few missing pixels, since it averages such

pixels with the other pixels in the frame. Furthermore, PSNR depends on the content of each video; for example, PSNR only penalizes low resolution if it is over a region with fine detail; rating streaming approaches reliably with a metric like PSNR requires testing over a large number of videos and user traces.

B. Trace-Based Experiment Results

We first evaluate CoRE and tiling with the AT&T network trace listed in Table III. This trace exhibits bandwidth fluctuation, especially from time 20 to 30 seconds, when there is significant bandwidth degradation. Thus, it represents a challenging case for both tiling and CoRE. We experiment with the remaining three network traces in §IV-B5.

1) Missing Pixels and Quality Changes: Figure 6 shows the percentage of frames missing at least one pixel (1MP) and Figure 7 depicts the maximum percentage of missing pixels (MMP) in any frame, when using the tiling variants that do not send all the tiles of a frame, i.e., FoV, FoV+ 1QL, and FoV+ 2QL. Since FoV+ 1QL and FoV+ 2QL use the same set of tiles (albeit at different quality levels), we only show the results for FoV+ 1QL. The error bars here and throughout the rest of the paper represent 95% confidence intervals (computed based on a normal distribution).

For FoV-only tiling in Figures 6 and 7, the Rhino video has the lowest 1MP and MMP values: $\sim$40% of the frames played have at least one pixel missing, and at least one frame misses as much as 81% of its pixels. The NY video has the highest 1MP of 60.3% and MMP of 95%. As expected, padding reduces the number of missing pixels, but does not eliminate them. For FoV+ 1QL and FoV+ 2QL tiling, we see 14(6)% 1MP and 51(33)% MMP for NY (Rhino) video in Figures 6 and 7. Therefore, with FoV-only tiling, a user is expected to notice missing pixels every two frames and will have at least one frame missing many pixels. For FoV+ 1QL and FoV+ 2QL tiling, the user will encounter missing pixels every five to ten frames and will have at least one frame missing about half its pixels. With CoRE, there are never any missing pixels.

We now quantify the changes in sampling rate for tiling with two quality levels. Figure 8 shows the % of frames with at least one pixel on the quality level change boundary (1AP, blue bars), and the maximum number of boundary pixels in a frame (multiplied by 100, labeled MAP and shown in red). Nearly half the frames have pixels on the quality change boundary, with as many as 1100 pixels in some frames. Abrupt changes in quality levels adversely affect user experience [2]. For CoRE, sampling rate changes smoothly.

2) Transfer Amounts: Figure 9 compares the amount of data transferred with tiling and CoRE. We observe that CoRE transfers less data than the FoV+ tiling variants in most cases, which saves network bandwidth and device energy. As expected, FoV-only tiling transfers the least amount among the tiling variants, followed by FoV+ 2QL, FoV 360, FoV+ 1QL, and FoV+ 360. The transferred data depends on the number of transferred tiles and the quality of the tiles. FoV 360 sends more tiles than FoV+ 1QL, but the latter has more higher quality tiles, making FoV+ 1QL more bandwidth-consuming than FoV 360. As expected, FoV+ 360 transfers the most data.

3) Frame Rates and Stalls: The means and confidence intervals of the frame rates and stalls for the six videos are shown in Figure 10. It is apparent from the figure that CoRE experiences little stalls and close to 30fps, while all tiling methods suffer from high stall durations and low frame rates. Among the tiling methods, FoV-only tiling transfers the least, and hence achieves better performance in terms of both average frame rate and stall duration. However, FoV-only tiling suffers from missing pixels (§IV-B1).

4) Low-Level Video Decoding Overhead: Once the client receives a video file from the server, the client has to decompress it with a standard video codec, e.g., H.264. The cumulative overhead of this low-level video decoding increases when a video is partitioned into smaller videos that are decoded individually. Figure 11 shows the video decoding overhead in terms of time and energy consumption. Timing data was collected on our laptop, whereas energy consumption was measured on our tablet by averaging the battery level drop from full, over all traces (Table II). CoRE, which requires decoding a single file every 4s, has a much smaller overhead than tiling, which requires decoding about 50 files for the FoV+ methods, and $6 \times 4 \times 4 = 96$ for the 360 methods. We conclude that CoRE provides significant decoding savings.

5) Impact of Network Conditions: We now use all network traces given in Table III. Due to space limitations, we only discuss the results for the Diving video, but we have experimented with the other videos, and find the results to be similar. We observe that the average sampling rates for CoRE are 62.18%, 62.19%, 65.33% and 66.69% for the four network bandwidth traces, respectively. Figure 12 compares the average frame rate and stall durations of all tiling methods and CoRE. We observe that CoRE outperforms all tiling variants.

We also experimented with different central and peripheral region sizes (1152×614 and 1323×1053), and with different extension lengths (6s and 9s). We find that enlarging the central region while reducing the peripheral region, keeping the CoRE frame dimensions the same, increases the average sampling rate, at the cost of increasing the file size. Increasing the extension part duration helps mask longer bandwidth drops, again, at the cost of a larger file. Note that the extension has only limited redundancy with the next chunk, due to the different predicted views and the decreasing frame rate.

6) Impact of View Direction Misprediction: We vary the view prediction accuracy from 100% to 60%. The prediction accuracy is computed from the degree difference between the predicted and actual user view directions. A prediction accuracy of x% implies that the predicted view direction is $\frac{180\times(100-x)}{100}$ and $\frac{90\times(100-x)}{100}$ degrees of longitude and latitude away from the user actual view direction.

We used the Verizon bandwidth trace (Table III) because its high bandwidth reduces the network effect on performance. We observe that sampling rate is correlated with prediction accuracy, e.g., reducing prediction accuracy from 100% to 60%, reduces the sampling rate by 38%, 42%, and 43% for

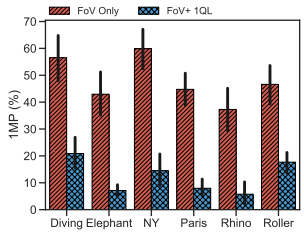


Fig. 6: 1MP (%).

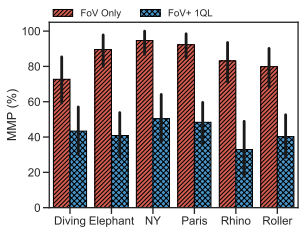


Fig. 7: MMP (%).

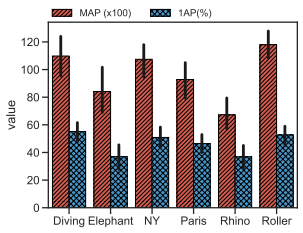


Fig. 8: 1AP and MAP.

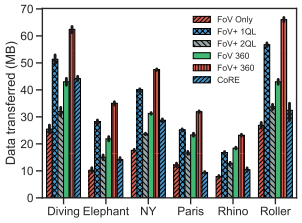
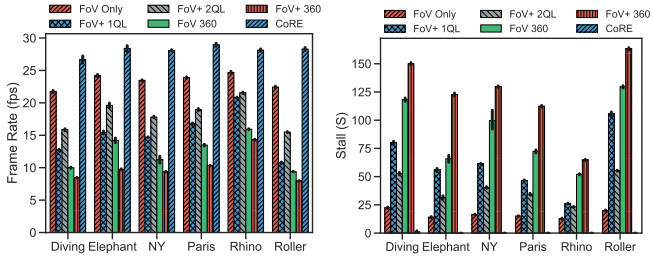
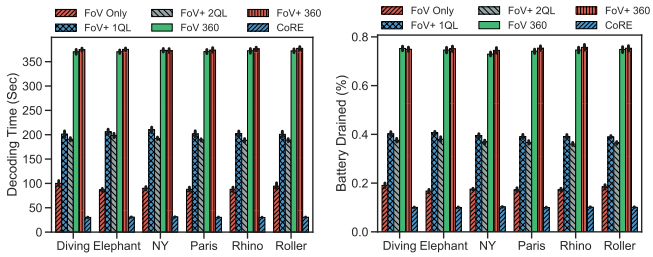


Fig. 9: Data transferred.



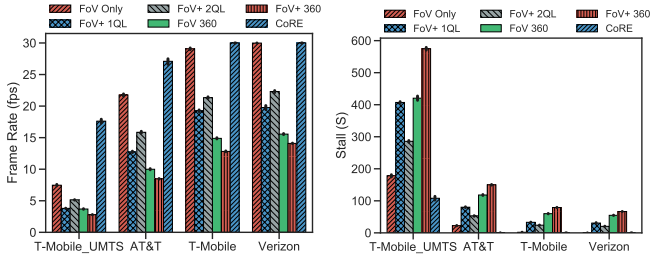
(a) Average Frame Rate (b) Stall Duration

Fig. 10: Frame rates and stalls with the AT&T bandwidth trace.



(a) Time (b) Energy

Fig. 11: H.264 video decoding overhead.



(a) Average Frame Rate (b) Stall Duration

Fig. 12: Diving video with four network bandwidth traces.

the Diving, Rhino, and Roller videos. Our experiments that use the current view direction without any prediction correspond to approximately 80% prediction accuracy.

C. User Study

We use our CoRE Android client prototype on the Samsung tablet described in §IV-A to conduct an IRB-approved user study using CoRE and FoV+ 1QL tiling. We use approximately two minutes each of three videos, streamed by a server running on the same laptop used in the above experiments. The

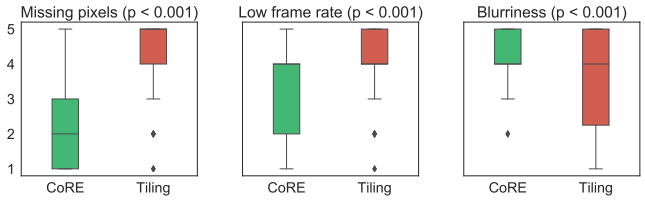


Fig. 13: Results of the user study.

client tablet screen can be swiped to change the view direction up, down, left, or right while watching the video. The tablet and laptop are connected via a hotspot created by a phone with AT&T 4G LTE service. The wireless network average download speed is approximately 1.03 MB per second (again we use Bytes for ease of comparison).

Thirty participants, ranging in age from 18 to 52, of both genders, watched and rated six videos (the Diving, Rhino and Roller videos in §IV-A2 with each of the two streaming methods). The order of the six videos is shuffled for each participant. We replay three swipe movement traces that we captured (one for each of the three videos) to generate the six videos that the participants rate. This has the benefit of giving all participants the same experience [6].

Figure 13 shows the participant ratings in terms of three problems: missing pixels, low frame rate (choppiness), and blurriness (blockiness) on a scale of 1 (strongly disagree) to 5 (strongly agree). As seen in the figure, CoRE outperforms (has lower average values than) tiling in terms of missing pixels and low frame rate, but tiling outperforms CoRE in terms of lower blurriness. The blurriness result is expected because CoRE avoids missing pixels by reducing peripheral sampling rates. For tiling, we are using the FoV+ 1QL tiling baseline, which uses a single resolution.

We now test the hypothesis that the distributions of responses for CoRE and tiling were equal across the three metrics. Given two populations P_1 and P_2, we evaluate the null hypothesis: $H_0 : P_1 = P_2$, and the alternate hypothesis, $H_1 : P_1 \neq P_2$, where P_1 and P_2 represent the observations for CoRE and tiling, respectively. Since the distributions are normal, we use a t-test [7] with a p value threshold $\alpha = 0.05$, to validate the null hypothesis. We find that for (1) presence of missing pixels ($p < 0.001$), (2) low frame rate ($p < 0.001$), and (3) blurriness ($p < 0.001$), the differences are statistically significant (Figure 13).

V. Related Work

The work most closely related to ours is Flare [17], Rubiks [8], Pano [6], ClusTile [29], POI360 [26], 360probDASH [25], Oculus360 [28], and streaming for 5G networks [19]. Flare [17] applies machine learning techniques for view direction prediction. Since prediction is imperfect, Flare sends additional tiles around the FoV, and develops scheduling algorithms and system optimizations to increase the output video quality. Rubiks [8] augments tiling with temporal segmentation into layers, which balances the decoding time, video quality and bandwidth consumption trade-off. Pano [6] uses variable-sized tiles– an idea proposed in ClusTile [29]. In POI360 [26], each video frame is spatially segmented into 96 tiles, scaled down based on their distances from the FoV center. The quality level is constant within a tile. 360probDASH [25] determines the tiles to pre-fetch and their quality level by solving a QoE-driven optimization problem. Sun *et al.* [19] send encoded video in multiple tiers over 5G. CoRE is inspired by ideas in these papers, but takes an alternative approach to tiling by gradually reducing the resolution, gradually reducing the frame rate, and storing an entire chunk in a single file. CoRE can incorporate several optimizations from the above work, as we discuss in §VI.

Oculus streaming uses a non-linear remapping of the 360° video based on a modified cube projection that allocates more pixels to the central region of the front face [28]. Unlike CoRE, the Oculus frame is discontinuous at the central line where it switches from the left-front-right to the top-back-bottom faces, and the resolution is fixed to one of three possible levels, lacking flexibility.

CoRE aims to increase robustness to transient bandwidth fluctuation using a decreasing frame rate extension. A complementary idea that can be integrated with CoRE is to predict available bandwidth, and to use it to decide the bit rate of the next chunks. Prior work on bit rate adaptation includes MPC [27], BOLA [18], and Pensieve [13] for stored video, and Jigsaw [1] and MPC-Live [20] for live video. These studies use control theory, optimization, machine learning, or layered encoding techniques to adapt the user quality of experience to available bandwidth and available buffer space. Several 360° video streaming approaches [6], [8], [17] have adopted MPC. Finally, non-linear sampling has long been studied in visualization [5], [11], [15], [16], [24]. Variable frame rate low-level encoding has been explored for videos where the frame infrequently changes [21].

VI. Discussion

Storage. The decreasing frame rate in the extension part of the CoRE chunk implies an increasing dissimilarity between consecutive frames and therefore less efficient compression (e.g., for the Diving video, the average extension frame size is 0.0265MB compared to 0.0156MB for the main part). Even so, an extension covering 6s beyond the 4s of the main part costs only about 30% of the total chunk size.

A CoRE chunk is constructed based on a predicted user view direction. One approach is for the server to construct the CoRE chunk on-demand, once it receives the predicted view direction. This scales poorly with the number of clients. Another approach, which we adopt, is to precompute and store CoRE chunks that cover all possible view directions. We use a 20° discretization of the horizontal and vertical directions: there are 18 chunks for the equator, and the number tapers off as latitude increases towards the poles, for a total of 105 CoRE chunks. The server transfers the CoRE chunk with the view direction that best matches the client request, using DASH. This approach scales well with the number of clients, at the cost of additional storage at the server. Storage can be reduced by only storing CoRE chunks for interesting regions of the video. For example, for the Roller video, only 15.6% of the CoRE chunks were requested in all traces in Table II.

CPU vs. GPU. CoRE is designed to support clients with no GPU: only a single video file has to be low-level decoded (§IV-B4), and the high-level CoRE decoding is optimized (§II-D). Tiling requires low-level decoding of tens of video files which precludes a CPU-only implementation (Figure 11a), hence the GPU focus of current work [6], [8], [17]. Should a GPU be present, CoRE can benefit from it for faster low-level decoding. On our Android device, low-level decoding a CoRE chunk on the GPU is $9.7\times$ faster than on the CPU; for tiling, the GPU advantage over the CPU is only $3.5\times$, as the number of tiles exceeds the GPU's number of hardware decoders. In terms of energy consumption, the GPU brings a $3\times$ advantage over the CPU for CoRE, and $1.1\times$ for tiling. We conclude that CoRE outperforms tiling on the GPU with an even larger margin than on the CPU.

Integration with other approaches. State-of-the-art 360° video streaming systems [6], [8], [17], [29] have proposed powerful optimizations that increase performance. Several of these optimizations can be incorporated into CoRE. For example, although the non-linear sampling in CoRE provides robustness to view prediction error, view trajectory prediction techniques, e.g., those employed by Flare [17], can be easily integrated with CoRE. Several of the Flare system optimizations across the protocol stack can also be incorporated into CoRE, such as request cancellation. Optimized low-level codecs as proposed in Rubiks [8] can also be used by CoRE. Bandwidth prediction and bit rate adaptation ideas from MPC [27], Flare [17], Rubiks [8], and Pano [6] can be incorporated into CoRE to select the bit rate of subsequent CoRE chunks to request from the server.

VII. Conclusions

We propose CoRE, an efficient approach for non-linear sampling in both the spatial and temporal domains that can increase the robustness of 360° video streaming to view direction prediction error and transient bandwidth fluctuation. Our experiments with a CoRE prototype demonstrate that CoRE saves bandwidth, with no missing pixels, and few stalls. CoRE does not require a GPU to achieve 30fps at the client. As future work, we plan to explore similar image generalizations for distributed applications that allow the client to not only change view direction, but also viewpoint.

REFERENCES

[1] BAIG, G., HE, J., QURESHI, M. A., QIU, L., CHEN, P., AND HU, Y. Jigsaw: Robust live 4K video streaming. In *Proceedings of MOBICOM* (2019).

[2] BEGEN, A., AND TIMMERER, C. Adaptive streaming of traditional and omnidirectional media. http://conferences.sigcomm.org/sigcomm/2017/files/tutorial-adaptive-streaming.pdf, 2017. ACM SIGCOMM 2017 Tutorial.

[3] CORBILLON, X., DE SIMONE, F., AND SIMON, G. 360-degree video head movement dataset. In *Proceedings of Multimedia Systems Conference* (2017), pp. 199–204.

[4] CORBILLON, X., DE SIMONE, F., AND SIMON, G. 360-degree video head movement dataset. http://dash.ipv6.enstb.fr/headMovements/, 2019.

[5] FURNAS, G. W. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1986), CHI '86, ACM, pp. 16–23.

[6] GUAN, Y., ZHENG, C., ZHANG, X., GUO, Z., AND JIANG, J. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of SIGCOMM* (2019).

[7] H. J. SELTMAN. Experimental design and analysis. http://www.stat.cmu.edu/ hseltman/309/Book/, 2019.

[8] HE, J., QURESHI, M. A., QIU, L., LI, J., LI, F., AND HAN, L. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of MobiSys* (2018).

[9] ISO. Dynamic adaptive streaming over HTTP (DASH). https://www.iso.org/standard/75485.html, 2019.

[10] KRISHNAN, S. S., AND SITARAMAN, R. K. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *Proceedings of the 2012 Internet Measurement Conference* (New York, NY, USA, 2012), IMC '12, ACM, pp. 211–224.

[11] LAMPING, J., AND RAO, R. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. In *Readings in Information Visualization*, S. K. Card, J. D. Mackinlay, and B. Shneiderman, Eds. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 382–408.

[12] MAHIMAHI PROJECT. Mahimahi wireless network traces. https://github.com/ravinet/mahimahi/tree/master/traces, 2019.

[13] MAO, H., NETRAVALI, R., AND ALIZADEH, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, ACM, pp. 197–210.

[14] PAXSON, V., ALLMAN, M., CHU, J., AND SARGENT, R. Computing TCP's retransmission timer. https://tools.ietf.org/html/rfc6298, June 2011.

[15] PIETRIGA, E., AND APPERT, C. Sigma Lenses: Focus-Context Transitions Combining Space, Time and Translucence. In *CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy, Apr. 2008), ACM, Ed., ACM, ACM.

[16] POPESCU, V., ROSEN, P., ARNS, L., TRICOCHE, X., WYMAN, C., AND HOFFMANN, C. M. The general pinhole camera: Effective and efficient nonuniform sampling for visualization. *IEEE transactions on visualization and computer graphics 16*, 5 (2010), 777–790.

[17] QIAN, F., HAN, B., XIAO, Q., AND GOPALAKRISHNAN, V. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of MOBICOM* (2018).

[18] SPITERI, K., URGAONKAR, R., AND SITARAMAN, R. BOLA: Near-optimal bitrate adaptation for online videos. In *Proceedings of IEEE INFOCOM* (April 2016).

[19] SUN, L., DUANMU, F., LIU, Y., WANG, Y., YE, Y., SHI, H., AND DAI, D. Multi-path multi-tier 360-degree video streaming in 5G networks. In *Proc. of the ACM Multimedia Systems Conference* (2018).

[20] SUN, L., ZONG, T., LIU, Y., WANG, Y., AND ZHU, H. Optimal strategies for live video streaming in the low-latency regime. In *Proceedings of IEEE ICNP* (2019).

[21] WAGGONER, B. *Compression for Great Video and Audio: Master Tips and Common Sense*. Taylor & Francis US, 2009.

[22] WANG, J., ZHENG, Y., NI, Y., XU, C., QIAN, F., LI, W., JIANG, W., CHENG, Y., CHENG, Z., LI, Y., XIE, X., SUN, Y., AND WANG, Z. An active-passive measurement study of TCP performance over LTE on high-speed rails. In *Proc. of ACM MOBICOM* (2019).

[23] WINSTEIN, K., SIVARAMAN, A., AND BALAKRISHNAN, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (Lombard, IL, 2013), pp. 459–471.

[24] WONG, N., CARPENDALE, S., AND GREENBERG, S. Edgelens: an interactive method for managing edge congestion in graphs. In *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No.03TH8714)* (Oct 2003), pp. 51–58.

[25] XIE, L., XU, Z., BAN, Y., ZHANG, X., AND GUO, Z. 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Proceedings of the 25th ACM International Conference on Multimedia* (New York, NY, USA, 2017), MM '17, Association for Computing Machinery, p. 315–323.

[26] XIE, X., AND ZHANG, X. POI360: Panoramic mobile video telephony over LTE cellular networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2017), CoNEXT '17, Association for Computing Machinery, p. 336–349.

[27] YIN, X., JINDAL, A., SEKAR, V., AND SINOPOLI, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, Association for Computing Machinery, p. 325–338.

[28] ZHOU, C., LI, Z., AND LIU, Y. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference* (New York, NY, USA, 2017), MMSys'17, Association for Computing Machinery, p. 27–37.

[29] ZHOU, C., XIAO, M., AND LIU, Y. ClusTile: Toward minimizing bandwidth in 360-degree video streaming. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications* (2018), p. 962–970.