Demo: Tile-Based Viewport-Adaptive Panoramic Video Streaming on Smartphones

Feng Qian[1*] Bo Han[2] Qingyang Xiao[1] Vijay Gopalakrishnan[2]

[1]Indiana University [2]AT&T Labs – Research

ABSTRACT

Flare is a practical system for streaming 360° videos on smartphones. It takes a viewport-adaptive approach, which fetches only portions of a panoramic scene that cover what a viewer is about to perceive. Flare consists of a novel framework for the end-to-end streaming pipeline, introduces innovative streaming algorithms, and brings numerous system-level optimizations. In our demo, we will show that Flare substantially outperforms traditional viewport-agnostic streaming algorithms in terms of the video quality. We will also invite the audience to use Flare to watch attractive 360° videos.

ACM Reference Format:

Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Demo: Tile-Based Viewport-Adaptive Panoramic Video Streaming on Smartphones. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3241539.3267715

1 INTRODUCTION

360° videos, also known as panoramic or spherical videos, are playing an important role in today's virtual reality (VR) ecosystem. 360° videos are recorded by omnidirectional cameras that are capable of capturing the whole panoramic scene. The panoramic scenes are then *projected* onto 2-dimensional frames using projection algorithms such as Equirectangular and Cube Map. During video playback, the player *reversely projects* each frame onto a 3-dimensional virtual sphere, with the viewer being in its center. Meanwhile, the viewer wearing a VR-headset can freely change her viewing direction, which, together with the headset's Field of View (FoV), determines the area (called *viewport*) visible to and thus rendered for the viewer.

360° videos provide highly immersive and interactive experiences to viewers. However, streaming 360° video contents over resource-constrained wireless networks is challenging. The key reason is that, due to their panoramic nature, 360° videos' sizes are much larger (4× to 6×) than regular videos under the same perceived quality. As a result, the player may need to download hundreds or thousands MB of data for streaming only a couple of minutes of 360° videos.

To reduce the high bandwidth consumption, a promising approach that is being actively studied by the research community is the *viewport-adaptive* streaming paradigm. Its high-level idea is straightforward: since the viewer only consumes a small fraction of a panoramic scene, the client player can prefetch the *visible area* based on predicting the viewer's future viewport. Areas that are not predicted to be consumed will not be downloaded, or they could be fetched at a lower quality. This approach can significantly reduce the bandwidth usage, or boost the video quality under the same bandwidth utilization compared to today's viewport-agnostic approach of downloading all 360° contents.

Despite the intuitive idea, developing a practical viewport-adaptive 360° video streaming system faces numerous challenges, just to name a few below. How to accurately predict a viewer's viewport? How to deal with inaccurate viewport prediction (VP)? How to play only a small area of the whole panoramic video? How to decide which area(s) to fetch? How to design a rate adaptation algorithm that selects the right video quality based on the current network condition and VP results? Finally, how to integrate everything into a holistic system running efficiently on smartphones?

To advance the state-of-the-art of 360° video streaming, we develop Flare, a practical viewport-adaptive streaming system running on smartphones. Compared to previous systems, Flare consists of a novel framework of the end-to-end streaming pipeline. Flare also introduces new streaming algorithms, and brings numerous system-level and network-level optimizations. In addition, Flare is a *general* framework for 360° video streaming that does not depend on a specific video encoding technology. Full technical details of Flare [2] will be presented at the main conference of ACM MobiCom 2018.

* Current affiliation: University of Minnesota – Twin Cities.

MobiCom '18, October 29-November 2, 2018, New Delhi, India

© 2018 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5903-0/18/10.

https://doi.org/10.1145/3241539.3267715

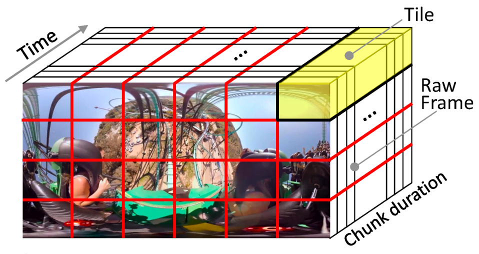

Figure 1: Chunk, tile, frame, and tile segmentation (4×6 tiles).

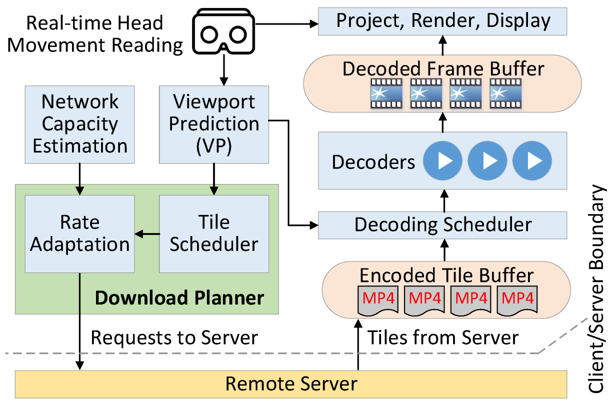

Figure 2: The Flare system architecture.

2 THE DESIGN OF FLARE

We now provide a high-level overview of the Flare system, which consists of a player application running on a commodity smartphone and a server application that hosts the 360° videos. All key logics including VP, rate adaptation, decoding, and rendering are performed on the client side.

We first describe how to prepare the video content. As shown in Figure 1, we segment each original panoramic video chunk into *tiles*. A tile (yellow area) has the same duration and number of frames as the chunk it belongs to, but occupies only a small spatial portion. Each tile can be independently downloaded and decoded. Therefore, ideally a player needs to download only tiles that cover a user's viewport trajectory.

Figure 2 sketches the high-level system design of Flare. The client performs VP in realtime to estimate the direction that the viewer is about to look at, using the head movement stream obtained from smartphone's sensors. Then a very important component on the client side is the *Download Planner*. It takes as streamed input the VP and network capacity estimation, and computes (1) the set of tiles to be downloaded and (2) their desired qualities, which are handled by *Tile Scheduler* and *Rate Adaptation*, respectively. In theory, these two aspects need to be jointly considered. But a key design decision we make is to *separate these two decision processes*, *i.e.*, first calculating the to-be-fetched tiles (the job of Tile Scheduler), and then determining their qualities (Rate Adaptation). The rationale behind this is two-fold. First, jointly considering both leads to a big inflation of the decision space and thus the algorithm complexity; second, (1) is more important than (2) because a missing tile will inevitably cause stalls. When tiles arrive from the server, they are properly buffered, decoded, projected, and rendered to the viewer, as shown in the right part of Figure 2. Compared to the client side, the server is relatively "dumb" – simply transmitting the tiles per clients' requests. This client-server function partition follows the DASH streaming paradigm, which facilitates scalability and ease of deployment. We next provide more details for the important components in Figure 2.

• **Viewport Prediction (VP).** We follow a typical online machine-learning (ML) paradigm by using the most recent *hw* (history window) seconds worth of head movement data to predict the viewport in the next *pw* (prediction window) seconds. This involves both training and testing. In our design, we take a very lightweight approach with the purpose of adapting to the continuous and fast-paced head movement. Specifically, Flare uses simple linear regression for $pw < 1$ and ridge regression (a variant of LR for better coping with overfitting) for $pw \geq 1$ given their reasonable accuracy.

• **Tile Scheduler** computes the set of tiles to be fetched based on VP. Assume that at time T_0, VP is invoked by the scheduler to update the to-be-fetched tile list. Instead of performing a single prediction, the scheduler conducts *multiple* predictions for time $t = T_0 + \delta_T, T_0 + 2\delta_T, ..., T_0 + m\delta_T$, in order to construct the *trajectory* of the user's future viewports. We pick δ_T=100ms and m=30, yielding a predicted trajectory of 3 seconds. Next, we map the trajectory to an *ordered list* of tiles. The higher rank a tile stands in the list, the more important the tile is, and the earlier the server will transmit the tile. When ranking the tiles, our algorithm jointly considers several aspects: each tile's distance to the center of the predicted viewport, the tile's playback time, and historical VP accuracy. The ordered tiles are then passed to the rate adaptation module to determine their qualities.

• **Rate Adaptation.** Flare's rate adaptation approach is inspired by MPC [3], which provides a principled model that considers different QoE objectives for Internet videos. Leveraging the high-level concept from MPC, we develop a practical formulation for rate adaptation of tile-based 360° video streaming. Our formulation considers four QoE objectives: consumed bitrate, stall duration, inter-chunk (temporal) quality change, and inter-tile (spatial) quality change. With these QoE objectives defined, the rate adaptation problem can be formulated as the following: determining the quality level for each to-be-fetched tile so that the overall utility, defined as a weighted sum of the four QoE objectives, is maximized. We then develop several performance boosting techniques that enable Flare to invoke the rate adaptation at a very fast pace at runtime to adapt to viewer's continuous head movement.

• **Dealing with Inaccurate VP.** Due to the randomness of human head movement, the VP is inherently imperfect. Flare employs three mechanisms to tolerate inaccurate VP. The first one is naturally provided by the tiles themselves: since a tile must be fetched as long as *any* of its frames intersects with *any* predicted viewport, oftentimes only a small portion of a fetched tile is consumed. This wastes some bandwidth but helps absorb inaccurate VP as long as the error does not cross a tile. The second mechanism is to fetch additional tiles that are not in the originally predicted tile set. We call them "out-of-sight" (OOS) tiles because if the prediction is accurate, a viewer will not see those tiles. The procedure of adding OOS tiles is integrated into the tile ranking algorithm performed by Tile Scheduler. The third mechanism is the server-side design with details described in our full paper [2].

• **Tile Decoding.** In Flare, a viewport typically contains multiple independently-encoded tiles that are played at the same time. We employ several techniques to effectively decode them. First, we leverage multiple concurrent hardware decoders (*e.g.,* 4 parallel H.264 decoders on SGS8) to accelerate the decoding process. Second, we allow decoders to cache the decoded (uncompressed) tiles to be played in the future into the *Decoded Frame Buffer* (DFB). Doing so improves the overall decoding performance and facilitates smooth playback when visible tiles change, as long as future tiles are properly cached in DFB. Third, we develop the *Decoding Scheduler*, which selects from the tiles waiting at the Encoded Tile Buffer (Figure 2) the most important ones to decode, in order to further mitigate decoding-incurred stalls.

Implementation and Evaluation. We have implemented Flare on commodity Android smartphones and Linux OS. The player is written in Java using Android SDK and C++ using Android NDK. Overall, our implementation consists of 14,200 lines of code (LoC). We have conducted extensive evaluations (~400 hours' playback on WiFi and ~100 hours over LTE). The experimental results demonstrate that Flare significantly improves the quality-of-experience (QoE) in real-world settings. Compared to non-viewport-adaptive approaches, Flare yields up to 18× quality level improvement on WiFi, and achieves considerable bandwidth reduction (up to 35%) and video quality enhancement (up to 4.9×) on LTE. Please refer to our full paper [2] for details.

3 DEMONSTRATION PLAN

Our demonstration testbed consists of three SGS8 smartphones as video clients (labeled as A, B, and C), two laptops as video servers (labeled as X and Y), one WiFi access point, and a VR headset. The smartphones and laptops are connected to our WiFi AP that provides local wireless connectivity. The servers are pre-loaded with several HD 360 videos in 4K resolution. The demonstration consists of two components: (1) comparing the video quality and resource consumption between Flare and a traditional viewport-agnostic streaming approach, and (2) letting the audience experience Flare in person. We next describe both components.

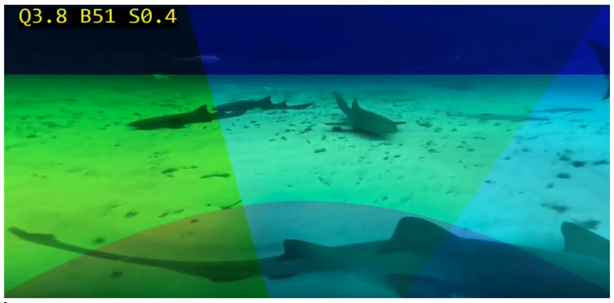

Figure 3: Our developed visualization module that renders tiles in different colors (9 tiles are shown). The three numbers on the top-left corner are key performance statistics: average quality level (Q), downloaded bytes (B) and cumulative stall duration (S).

Performance Comparison. We put smartphone A and B side-by-side, and use laptop X as their server. Phone A runs the Flare system, and Phone B runs a classic viewport-agnostic algorithm such as BBA [1]. Both smartphones will replay the same head movement trace collected from a real user. On the server side, we emulate diverse network conditions by using the `tc` tool to replay pre-recorded bandwidth traces collected from real cellular networks. The audience will be able to tell the video quality difference between Flare and BBA, with Flare being better. In addition, from the performance statistics shown on the screen (Figure 3), the audience can also quantitatively compare their key metrics including average quality, bandwidth usage, stall duration.

In-person Experience. We insert Smartphone C into a VR headset and use laptop Y as the server. We invite the audience to watch 360° videos on the headset. To visualize the tiles, we develop an optional visualization module that renders tiles in different colors, as shown in Figure 3. The viewer can try different videos as well as change configurations such as the segmentation scheme (4×6 tiles by default), VP algorithm, and the number of parallel decoders.

Our demo requires the default space (one 6 × 2.5 ft table) and three power outlets. No Internet connectivity is needed as we use a local WiFi network provided by our AP. The expected setup time of our demo is less than 5 minutes.

REFERENCES

[1] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of SIGCOMM 2014*, pages 187–198. ACM, 2014.

[2] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *ACM MobiCom*, 2018.

[3] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of SIGCOMM 2015*, pages 325–338. ACM, 2015.