STC: Enabling 16K VR streaming on mobile platforms with FoV tracking

Conference Paper · December 2020

DOI: 10.1109/GLOBECOM42002.2020.9322384

5 authors, including:

Yu Guan
Peking University
15 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Xinggong Zhang
Peking University
69 PUBLICATIONS **834** CITATIONS
SEE PROFILE

Zongming Guo
Yantai Nanshan University
211 PUBLICATIONS **3,523** CITATIONS
SEE PROFILE

STC: Enabling 16K VR streaming on mobile platforms with FoV tracking

Chengyuan Zheng, Jinyu Yin, Yu Guan, Xinggong Zhang, Zongming Guo
Wangxuan Institute of Computer Technology, Peking University
Beijing, P.R. China
zhengchengyuan@pku.edu.cn, yinjinyu@pku.edu.cn, shanxigy@pku.edu.cn, zhangxg@pku.edu.cn, guozongming@pku.edu.cn

***Abstract*— 16K VR videos are coming to ages. But it could overwhelm mobile hardware for its huge bandwidth consumption and decoding complexity. To enable 16K VR video streaming over mobile platforms, we present a novel ShiftTile-Tracking (STC) streaming system, which crops and transmits video by tracking the Field-of-View (FoV) movement of users. The video chunk is split into ShiftTiles with frame granularity, which always covers FoV areas along the FoV movement trajectory. This transforms a 360-degree VR video into a traditional planar video, which leads to huge bandwidth saving and faster decoding speed. In the system design, we mainly entail two contributions. 1) To accommodate various FoV movement trajectories with a limited number of ShiftTiles, we proposed an optimal tiling algorithm by FoV trajectory clustering. 2) To be resilient to the FoV prediction errors, we propose an accuracy-sensitive streaming algorithm, which expands the FoV area if the FoV prediction errors are high. The evaluation shows that under the same real-world 4G network conditions, the proposed STC improves 0.9dB V-PSNR, reduces 12.4% buffering ratio, and achieves 45% faster decoding speed (64 frames per second) on average compared with the state-of-the-art solutions. This enables 16K VR video streaming on current mobile platforms.**

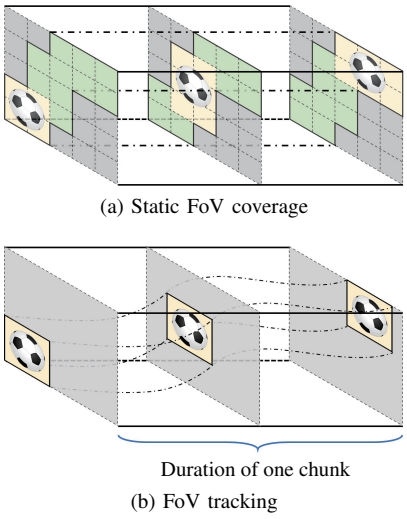
***Index Terms*—Virtual Reality, Adaptive Video Streaming, Tiling, Mobile Platforms**

I. Introduction

Virtual Reality (VR) video applications are booming. According to a survey of quartz [1], there will be 55 million mobile VR users in 2021, and mobile VR devices (e.g., VR glasses [2] or head-mounted devices [3]) will account for 80% of VR platforms. However, most of today's deployed VR video systems only support 4K resolution, which is equivalent to display a planar 240P video on desktops. This would make the user suffer from VR sickness [4]. To provide a better Quality-of-Experience (QoE), 16K VR videos (*e.g.,* 15360 × 7680) are required which can achieve 40 Pixel-Per-Degree (PPD) with ultra-high-definition quality [5].

However, streaming 16K VR videos on mobile platforms is a great challenge. It needs huge **bandwidth consumption** and **formidable computation resources**. (1) One encoded 16K VR video consumes around 200Mbps to 1.65Gbps bandwidth [6], which is a huge burden even for the state-of-the-art 5G network [7]. (2) Today's mobile CPU can only decode 8K (*e.g.,* 7680×3840) videos at the most. It is hard to display 16K VR videos on mobile platforms, let alone decode them at 60 frames per second (fps).

FoV adaptive (or called viewport adaptive) streaming is a well-known solution to deliver VR video with less bitrate [8]. It first maps VR videos into equal-rectangular planar videos [9], and then split them into grid-like tiles [10], [11] in the spatial dimension. By FoV prediction [8], [12], one only needs to stream the tiles overlapped with the user's FoV, thus significantly reduces the computation cost and bandwidth consumption. But it is still tough to stream 16K videos with the FoV adaptive methods since they cover FoV areas with static grid-like tiles, which is called static FoV coverage. As shown in Figure 1 (a), a tile will be delivered entirely even only one frame is inside the FoV trajectory. It would lead to severe bandwidth waste especially when FoV is moving fast.



(a) Static FoV coverage

(b) FoV tracking

Figure 1: A high-level comparison between static FoV coverage and FoV tracking solutions. Yellow areas are transmitted and viewed by users and green areas are transmitted but not viewed by users. Suppose the user is looking at the moving football in one video chunk.

Unlike the aforementioned methods, we try to stream VR videos in an FoV tracking way. It crops FoV areas along FoV movement trajectories, named **ShiftTiles**, which are delivered and displayed like an ordinary 2D video. Figure 1 shows a comparison between the FoV tracking solution and the static FoV coverage solution. We can see that the FoV tracking solution saves lots of pixels compared with the other. It is equivalent to encoding and streaming one traditional 2D video with a limited viewing angle, thus it can significantly reduce

bandwidth consumption and decoding complexity.

However, it is not trivial to design a VR streaming system with FoV tracking. There are two technical challenges:

- Users' FoV movement is random. There may exist thousands of FoV trajectories for the same video. It is impossible to prepare ShiftTiles for all possible trajectories.
- There exist FoV prediction errors. Since the ShiftTiles are delivered by the predicted FoV trajectory, the accuracy of FoV prediction has great impacts on streaming quality.

To explore the opportunities above, we present ShiftTile-Tracking (STC), a novel VR video streaming system with the FoV tracking method. It generates ShiftTiles according to users' FoV trajectories. *STC* mainly entails two contributions. (1) To reduce the storage overhead, *STC* first utilizes an FoV dataset and groups similar FoV trajectories into FoV trajectory clusters. Then we propose a trace-driven optimization problem to cover them with a limited number of ShiftTiles (§II). (2) To be resilient to FoV prediction errors, we propose an accuracy-sensitive streaming algorithm. We make a data-driven analysis to estimate the FoV prediction accuracy. And we expand streaming areas (*e.g.,* more ShiftTiles) when the FoV prediction is uncertain (§III).

The evaluation shows that under the same 4G network conditions, the proposed STC improves 0.9dB V-PSNR, reduces 12.4% buffering ratio, and achieves 45% faster decoding speed (64 frames per second) on average compared with the state-of-the-art solutions. This enables 16K VR video streaming on today's mobile platforms.

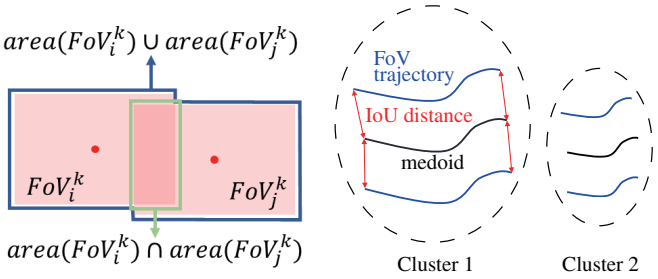
The rest parts of the paper are organized as follows. We present the optimal tiling algorithm in §II. Then to provide robust streaming against FoV prediction errors, we introduce an accuracy-sensitive streaming algorithm in §III. We evaluate STC with both trace-based simulation and real-world decoding evaluation in §IV. Finally, we summarize the paper and conclude the evaluation results in §V.

II. ShiftTiling

In this section, we present how we use a limited number of ShiftTiles, which crops FoV areas along FoV movement trajectories, to ensure every possible FoV trajectory can be covered. We found that FoV trajectories of different users have a strong similarity and we cluster the similar historic FoV trajectories (§II-A). Then we design the ShiftTile and StaticTile, and we establish a tiling optimizing problem to obtain the best parameter settings for them (§II-B).

A. FoV trajectories clustering

We cluster the users' FoV trajectories into a limited number of clusters according to the distance between them during one video chunk. Without clustering, the server needs to encode one specific video for each user's FoV trajectory, which will lead to severe computation and storage overhead. Besides, lots of previous studies [11], [12] have proved the FoV trajectory similarities between users in the same video event (*e.g.,* playing basketball and skating).



(a) Calculating IoU of two FoVs (b) K-medoids clustering

Figure 2: A sketch graph of FoV trajectory clustering

First, we define the Intersection-over-Union [13] (IoU) distance D_{IoU} between two FoVs in each frame. Figure 2 (a) shows a sketch graph of IoU between two FoVs. FoV_i^k and FoV_j^k denote the FoVs from the ith user and jth user in the kth frame correspondingly. Therefore the IoU distance D_{IoU} between FoV_i^k and FoV_j^k can be expressed as

$$D_{IoU}\left(FoV_i^k, FoV_j^k\right) = 1 - \frac{area(FoV_i^k) \cap area(FoV_j^k)}{area(FoV_i^k) \cup area(FoV_j^k)} \tag{1}$$

Second, we define the trajectory distance D_{tra} between two FoV trajectories during one video chunk (*e.g.,* 60 frames each chunk). D_{tra} is the average D_{IoU} each frame between two FoV trajectories in the whole video chunk, whose frame number is f. tra_i and tra_j are the FoV trajectories from the ith and jth users. FoV_i^k and FoV_j^k denote the FoVs in the kth frame from tra_i and tra_j. Thus we obtain the trajectory distance D_{tra} between tra_i and tra_j as

$$D_{tra}\left(tra_i, tra_j\right) = \frac{\sum_{k=1}^{f} D_{IoU}\left(FoV_i^k, FoV_j^k\right)}{f} \tag{2}$$

Third, K-medoids [14] algorithm is applied to cluster all the FoV trajectories based on the D_{tra}. Figure 2 (b) shows a sketch graph of K-medoids FoV trajectories clustering.

B. Optimal ShiftTiling of clustered FoV trajectories

Given a proper clustering algorithm, then we require an appropriate tiling method. The straightforward thinking is encoding the areas of all FoV trajectories in this cluster together into a ShiftTile. However, this is inefficient in practice, because this area is still 1.5× to 4× larger than a single user's FoV. Transferring this area to a user will lead to a serious waste of bandwidth.

To avoid this bandwidth waste, given an FoV trajectory cluster, we set a popularity threshold L_n for the nth FoV trajectory cluster. In this cluster, only the pixels viewed by more than L_n historic users will be encoded into one ShiftTile.

Obviously, the ShiftTile is not necessary to cover all historic FoV trajectories, and also there is no guarantee that they can match all users in the future. When a user's FoV trajectory

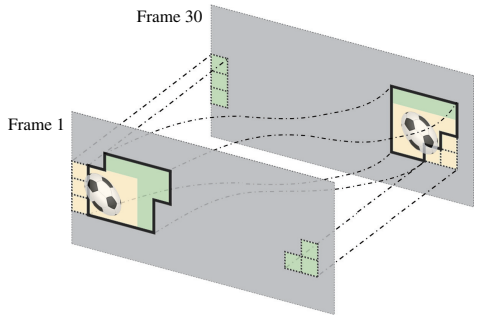


Figure 3: An example of how StaticTiles and ShiftTiles together cover a moving FoV trajectory. The yellow areas are transmitted and viewed by users and the green areas are transmitted but not viewed by users. The bounding box with solid lines means the ShiftTile and that with dotted line means the StaticTile.

can not be covered by any of the encoded ShiftTiles, there will be a serious depression of Quality-of-Experience (QoE).

To solve the problem, we encode each video chunk into $X \times Y$ equal-sized static rectangular tiles as the supplement of ShiftTile. We call it **StaticTile** in this paper. X and Y are parameters of the StaticTiles granularity (*e.g.,* 6×12, 12×24). If the user's FoV trajectory can be fully covered by one ShiftTile, we allocate the ShiftTile to her. Otherwise, we allocate a ShiftTile and some supplemental StaticTiles to her to guarantee the full coverage of her FoV trajectory. Figure 3 shows an example of how ShiftTiles and StaticTiles cover a user's moving FoV together.

Given that we first encode the video chunk into several ShiftTiles (the number is equal to the number of trajectory clusters) and then we encode the video chunk again into fine-grained StaticTiles as a supplement, how to put them together and make the best parameter settings is a problem. In the parameter setting, there are two important trade-offs.

The size of ShiftTiles. When we decrease the value of L_n, the ShiftTile will be larger, thus we need fewer StaticTiles to cover the remaining FoV. However, if the ShiftTile is too large, it may contain more areas outsides the user's FoV, leading to a waste of bandwidth.

The granularity of StaticTiles. It is well studied that when we spatially split a video chunk with fine granularity, it significantly lowers the encoding efficiency (*e.g.,* the total size of tiles will be much larger than the original video chunk). On the other hand, when we split a video chunk with coarse granularity, it leads to a waste of bandwidth: (1) It may have more overlap with the ShiftTile. (2) It may contain more areas outsides the user's actual FoV.

Our goal is to cover the user's FoV trajectory by ShiftTiles and StaticTiles with minimal expected bandwidth consumption. Given a video chunk, assume that we have clustered the FoV trajectories into K clusters, and $T_n(L_n)$ denotes the ShiftTile of the nth cluster with popularity threshold L_n (as described above, one trajectory cluster with a popularity threshold uniquely defines a ShiftTile). (X, Y) is the granularity of StaticTiles. Then we optimize the parameters $< L_1, ..., L_K, X, Y >$ by the following optimization problem:

$$\begin{aligned} \text{minimize:} \quad & \frac{1}{|U|} \sum_{u \in U} F_u(L_1, ..., L_K, X, Y) \\ \text{subject to:} \quad & (X, Y) \in \{(6, 12), (12, 24), (15, 30)\} \\ & L_n \in \{0\%, 20\%, 40\%, ..., 100\%\} \quad 1 \leq n \leq K \end{aligned} \tag{3}$$

where U denotes the set including all users in our dataset. $F_u(L_1, ..., L_K, X, Y)$ is the bandwidth consumption to cover the FoV trajectory of user u by ShiftTiles $T_1(L_1)$, ..., $T_K(L_K)$ and $X \times Y$ StaticTiles. Since the parameter space (18 combinations) of this optimization problem is small enough, we directly make an enumeration to solve it.

III. Accuracy-sensitive streaming algorithm

In this section, we propose an accuracy-sensitive streaming algorithm to be resilient to FoV prediction errors. According to our data analysis, although it is difficult to accurately predict the FoVs, the FoV prediction accuracy can be well-estimated (§III-A). So in the client-side tile selection, we adaptively expand the streaming areas based on the estimated FoV prediction accuracy (§III-B).

A. Data-driven FoV prediction analysis

We first obtain 4200 traces of FoV predictions by using the cross user method [12] to predict the FoV trajectories in the public dataset [15].

Then based on 4200 traces of FoV predictions, Figure 4 shows that the FoV prediction accuracy is positively correlated to the similarity between the current FoV trajectory and its closest historic trajectory cluster. The similarity between FoV trajectories is defined as subtracting the D_{tra} from one. For example, if the current FoV trajectory has a dominantly higher similarity with one historic FoV trajectory cluster than other clusters, there is high confidence that the prediction is reasonable. Otherwise, if the current FoV trajectory stands between multiple FoV trajectory clusters, there is a high probability that the prediction will fail. This is because when an incoming user acts similarly with many history users, her future action can be well-estimated by lots of historic data.

The finding gives us an insight that we can dynamically adjust our streaming areas based on the estimated prediction accuracy. For example, we can expand the streaming areas to the user when FoV predictions are likely to fail.

B. Adjusting streaming areas based on FoV prediction accuracy

In the client-side streaming logic, we adaptively adjust the size of FoV coverage areas according to the similarity between current FoV trajectory and historic FoV trajectory clusters. Based on 4200 historic traces of FoV predictions, Figure 5 shows that given a current FoV trajectory, when its highest similarity with historic clusters is over 70%, covering one

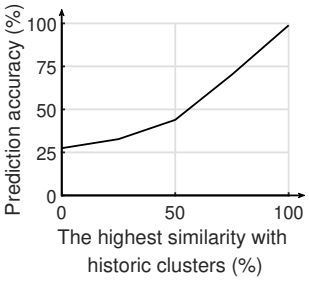


Figure 4: Correlation between similarity with historic clusters and predicting accuracy.

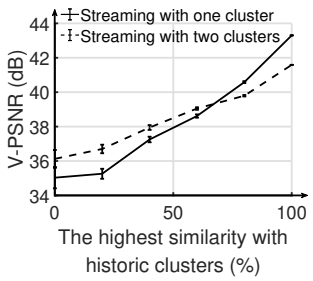


Figure 5: V-PSNR of different streaming algorithms under the same fixed bandwidth.

possible future trajectory is better. When it is below 70%, covering two is better.

Then based on the adaptive FoV coverage methods, we run an optimization problem in the client-side logic. The optimization problem is to maximize the video quality V-PSNR [16] of selected ShiftTiles and StaticTiles with the restriction of a given bandwidth consumption S and full coverage of one or two possible FoV trajectories. Assume P is the set of pixels in the most possible one or two FoV trajectories. A is the set of all available tiles for the video chunk (including different quality versions of ShiftTiles and StaticTiles). The client-side logic needs to select a part of them, called A' ($A' \subset A$), to cover the trajectories within the given bandwidth S. $Y(A')$ denotes the V-PSNR of selected quality versions of ShiftTiles and StaticTiles. $B(A')$ denotes their bandwidth consumption, and $E(A')$ is the pixels inside them. Then we make the client-side tile selection by the following optimization problem:

$$\begin{aligned} \text{maximize;} \quad & Y(A') \\ \text{subject to:} \quad & A' \subset A \\ & B(A') \leq S \\ & P \subset E(A') \end{aligned} \tag{4}$$

To solve this problem, enumerating all possible combinations is impractical because the number of different combinations could reach $2^{|A|}$ (In our experiments, the value of $|A|$ is usually around 1500). Thus we derive a two-step solution to reduce computation complexity.

First, we only ensure the full coverage of users' FoVs. And the coverage is not relative to the quality versions. Once the selection of ShiftTiles is determined, the optimal selection for StaticTiles is straightforward: because one pixel is covered by only one StaticTile, we just select each pixel in the FoV trajectory that is not covered by the ShiftTiles and cover it by the corresponding StaticTile. Typically, there exist 5 ShiftTiles of the same quality version in our algorithm. Thus we can obtain 2^5 combinations to cover the users' FoVs.

Second, we satisfy the bandwidth constraint and obtain the final results. Once given a specific FoV coverage solution, the problem can be transformed into a Multiple Choice Knapsack (MCK) problem [17]. There exist several tiles of different positions that are used for FoV coverage. Each tile has several quality versions with their bandwidth consumption and V-PSNR. The MCK problem is how to select the quality versions of different tiles to maximize the total V-PSNR within the bandwidth constraint. Since we have 2^5 FoV coverage solution, we solve 2^5 MCK problems and take the maximum of solutions as the results. Owing to the high-efficiency solutions to the MCK problem [18], the MCK problems can be solved within 0.1s on mainstream mobile platforms.

IV. EVALUATION

In this section, we evaluated *STC* with both trace-based simulation and real-world decoding evaluation.

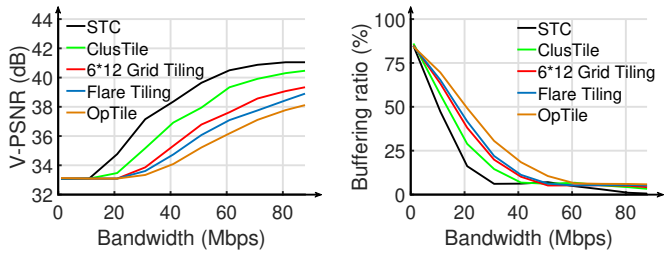
A. Methodology

Dataset: We use 16 VR videos [15] (618 seconds in total) with actual 34 user FoV trajectories (aged 20 to 24). The genres of videos are mainly sports. For specifically, due to the lack of 16K original videos in current VR video datasets, we up-sampling the original VR video onto the corresponding 16K videos. And each video is encoded into 5 quality levels (QP=22, 27, 32, 37, 42) and 1-second chunks using the FFmpeg [19] and x264 [20]. We also obtain real 4G traces in [21] to run the trace-based evaluation.

Experiment environment: We randomly choose FoV trajectories from 26 users that are used for running the tiling optimization problem (*e.g.,* proposed *STC*). And FoV trajectories of the other 8 users are used for evaluations. In trace-based simulations, we build a VR video mock-up system of dash on C# and Matlab. In real-world decoding evaluations, we build a test-bed based on the Unity pro [22] and ExoPlayer [23] with mediacodec. As for hardware platforms, we choose a powerful Windows Server 2016-OS desktop (Intel Xeon E5-2620v4 CPU, 160GB RAM) as the video provider to run the trace-based simulation experiment. And mobile platform vivo iqoo (Qualcomm Snapdragon 855 CPU, 12GB RAM) is used to run the decoding evaluation.

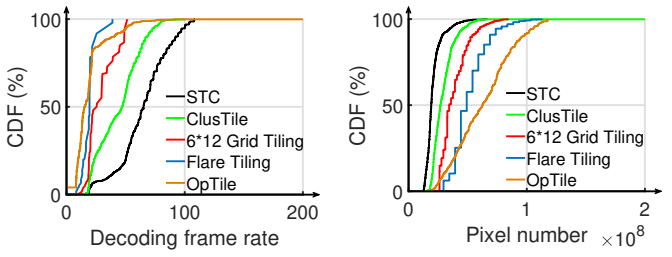
Baselines: We compare *STC* with four proposals, OpTile [10], ClusTile [11], Grid-like Tiling [12], and Flare [24]. Flare and Gird-like tiling split the video chunk into 4×6 and 6×12 rectangular StaticTiles. OpTile split the video chunk into different sizes of tiles according to the encoding efficiency. ClusTile is the state-of-the-art tiling scheme, which splits the video chunk into different sizes of tiles according to both the FoV distribution and encoding efficiency. Since OpTile and ClusTile are not open-source projects, we implement them strictly according to [10], [11] and optimize all parameter settings to reach their best performance. We set the grouping clusters of both *STC* and ClusTile to 5. For a fair comparison, all baselines and *STC* use the same logic for FoV prediction (cross-user FoV prediction [12]) and the same bitrate adaptation algorithm.

Video quality metrics: In our evaluation, We evaluate the video quality with V-PSNR [16], buffering ratio, and frame rate, which are critical to user experience. V-PNSR is a typical evaluation metric in VR video quality assessment. And the



(a) Bandwidth - V-PSNR (b) Buffering ratio - V-PSNR

Figure 6: Bandwidth - V-PSNR and Buffering ratio - V-PSNR trade-off under fixed bandwidth



(a) Decoding frame rate (b) Decoded pixel number each frame

Figure 8: Cumulative distribution graph of decoding frame rate and decoded pixel number each frame

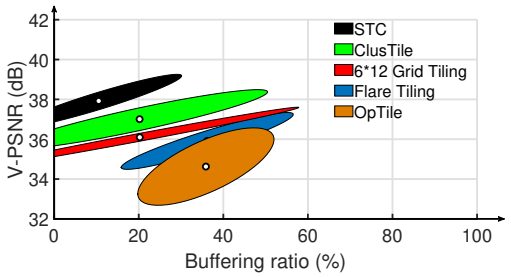


Figure 7: Buffering ratio - V-PSNR trade-off under fluctuating bandwidth

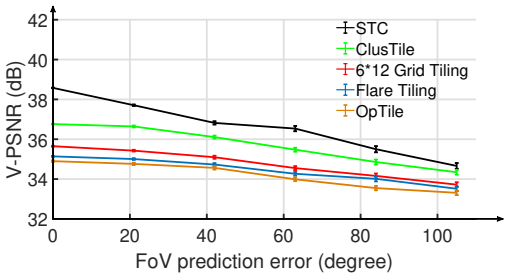


Figure 9: V-PSNR under different FoV prediction errors

buffering ratio in our evaluation is defined as the stalling time during the whole playback. Frame rate is the final video display rate which is influenced by the decoding speed.

B. Fixed and fluctuating bandwidth evaluation

In the fixed bandwidth evaluation, We apply 10 fixed bandwidth and the same bitrate adaptation algorithm to all the baselines and proposed *STC*.

Under the same fixed bandwidth, Figure 6 shows that *STC* achieves 1.2dB higher V-PSNR and 10.2% less buffering ratio compared with the state-of-the-art solution. In *STC*, our ShiftTiles track the FoV movement more accurately and save more pixels outside users' FoVs, which leads to more bandwidth saving and reduce the possibility of buffering.

In the fluctuating bandwidth evaluation, we pick the real 4G traces [21] as the input bandwidth. For specifically, we apply different bitrate adaptation parameters to all the methods at the same time to control the aggression. For example, when the bitrate adaptation is getting more aggressive all the methods will tend to have a higher buffering ratio and V-PSNR.

Compared with the state-of-the-art solution, Figure 7 shows that *STC* achieves 0.9dB higher V-PSNR and 12.4% less buffering ratio on average than the state-of-the-art solution with the same real 4G traces.

C. Real-world decoding evaluation

In the decoding evaluation, Figure 8 (a) shows that *STC* achieves 64 fps, which is 45% faster than the state-of-the-art solution. For a fair comparison, we only use the hardware decoders and guarantee the same utilization of them on mobile platforms across all the baselines and *STC*. Our faster decoding speed is owing to our huge pixel saving from ShiftTiling, and Figure 8 (b) shows that *STC* reduces 29% decoded pixel number each frame compared with the state-of-the-art solution.

D. Robustness against FoV prediction errors

To evaluate *STC*'s robustness against FoV prediction errors, we simulate the FoV prediction with different levels of errors and apply these predicted FoV trajectories to *STC* and baselines. All the methods are evaluated with the same fixed bandwidth. Considering that FoV prediction errors have a significantly bad influence on V-PSNR, the variation of V-PSNR under FoV prediction errors is presented in this section.

Figure 9 shows that *STC* achieves substantial 0.3dB to 1.8dB higher V-PSNR quality than the state-of-the-art solution with an increasing level of FoV prediction errors. Under the low prediction error, *STC* streams fewer pixels for areas not viewed by users, saving bandwidth to raise the quality of areas that are more likely to be viewed. And under the high prediction error, the accuracy-sensitive streaming algorithm expands the streaming areas. Although this increases the bandwidth consumption, it avoids the cases that the actual FoV is not covered by any tile, which leads to more severe V-PSNR depression. So this makes *STC* perform better than the baselines when there are substantial FoV prediction errors.

E. System overhead

Next, we examine the system overheads of the proposed *STC* and baselines on storage overhead and server-side preprocessing time under a one-minute video. All the baselines

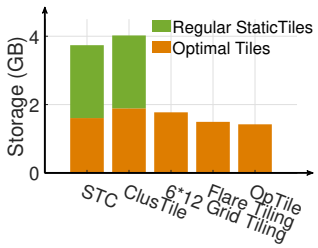


Figure 10: Storage comparison of a one-minute video

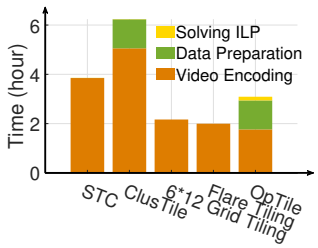


Figure 11: Preprocessing time comparison of a one-minute video

and *STC* use the standard FFmpeg and x264 to encode the corresponding tiles in the same parameter settings.

Server-side storage overhead: Figure 10 shows that *STC* saves 7% storage than ClusTile. For specifically, to prevent the difference between the historic users and incoming users, both *STC* and ClusTile need to store a regular StaticTiles (*e.g.,* 12×24 grid-like tiles), which leads to 2.64× and 2.82× bigger storage than OpTile. If we assume the consistency between historic users and incoming users, *STC* will only lead to 1.17× bigger storage than OpTile. This is because we encode only one ShiftTile and corresponding supplementary StaticTiles for each FoV trajectory cluster. So most of the pixels are covered by only one ShiftTile and few supplementary StaticTiles.

Server-side preprocessing overhead: Figure 11 shows that *STC* saves 36.1% preprocessing time than ClusTile. In *STC*, due to few number of tile combinations, ILP solving only accounts for less than 1% preprocessing time. And *STC* needs to encode 6 popularity of ShiftTiles each cluster, which are used to solve the tile optimization problems. ClusTile encodes 10 tiles of each cluster and has a longer encoding time than *STC*. ClusTile also needs to predict the bitrate of all combined StaticTiles using huge data of motion vectors, which leads to severe time consumption of data preparation.

Overall, we think that the server-side storage and video processing overhead of *STC* are still acceptable. It is not trivial considering its buffering ratio reduction, V-PSNR improvement, and client-side decoding acceleration.

V. Conclusion

In 16K VR video streaming, prior solutions design systems in the way of static FoV coverage. A tile will be delivered entirely even only one frame is inside the FoV trajectory, limiting the room for improving VR video streaming quality especially when the user's FoV is moving.

In contrast, we show that it is possible to enable 16K VR video streaming in an FoV tracking way. We encode the video into ShiftTiles which crops FoV area of each frame along the FoV movement trajectory. And to be resilient to FoV prediction errors, we propose an accuracy-sensitive streaming algorithm, which expands streaming areas (*e.g.,* more ShiftTiles) when the FoV prediction is uncertain. Our evaluation shows that under the same real 4G bandwidth traces, *STC* improves 0.9dB V-PSNR, reduces 12.4% buffering ratio, and achieves 45% faster decoding speed (64 frames per second) on average compared with state-of-the-art solutions on today's mainstream mobile platforms.

In conclusion, *STC* verifies that it is practical to enable 16K VR video streaming of 60 fps on today's mobile platforms.

References

[1] In five years, vr could be as big in the us as netflix - quartz. [Online]. Available: https://qz.com/1298512/vr-could-be-as-big-in-the-us-as-netflix-in-five-years-study-shows/

[2] Huawei vr glass. [Online]. Available: https://consumer.huawei.com/cn/wearables/vr-glass/

[3] Oculus quest. [Online]. Available: https://www.oculus.com/quest/?locale=zh_CN

[4] L. Clift, "Novel control systems for virtual reality, and their effects on cyber sickness."

[5] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva, "Vr is on the edge: How to deliver 360 videos in mobile networks," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, ser. VR/AR Network '17, 2017, pp. 30–35.

[6] Vanilla 16k uhd future tech. [Online]. Available: https://vanillavideo.com/features/specifications/16k/

[7] 5g speed: 5g vs 4g performance compared. [Online]. Available: https://https://www.tomsguide.com/features/5g-vs-4g

[8] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *The Workshop on All Things Cellular: Operations*, 2016, pp. 1–6.

[9] Z. Chen, Y. Li, and Y. Zhang, "Recent advances in omnidirectional video coding for virtual reality: Projection and evaluation," *Signal Processing*, vol. 146, pp. 66–78, 2018.

[10] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "Optile: Toward optimal tiling in 360-degree video streaming," in *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017, pp. 708–716.

[11] C. Zhou, M. Xiao, and Y. Liu, "Clustile: Toward minimizing bandwidth in 360-degree video streaming," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 962–970.

[12] L. Xie, X. Zhang, and Z. Guo, "Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming," in *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 2018, pp. 564–572.

[13] Jaccard index - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index

[14] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[15] Y. Xu, Y. Dong, J. Wu, Z. Sun, Z. Shi, J. Yu, and S. Gao, "Gaze prediction in dynamic 360 immersive videos," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5333–5342.

[16] M. Yu, H. Lakshman, and B. Girod, "A framework to evaluate omnidirectional video coding schemes," in *2015 IEEE International Symposium on Mixed and Augmented Reality*, Sep. 2015, pp. 31–36.

[17] P. Sinha and A. A. Zoltners, "The multiple-choice knapsack problem," *Operations Research*, vol. 27, no. 3, pp. 503–515, 1979.

[18] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, and L. Tassiulas, "Caching and operator cooperation policies for layered video content delivery," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[19] Ffmpeg. [Online]. Available: http://ffmpeg.org

[20] x264, the best h.264/avc encoder - videolan. [Online]. Available: https://www.videolan.org/developers/x264.html

[21] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: a 4g lte dataset with channel and context metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 460–465.

[22] Unity. [Online]. Available: https://unity.com/

[23] Exoplayer. [Online]. Available: https://github.com/google/ExoPlayer

[24] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 99–114.