Firefly: Untethered Multi-user VR for Commodity Mobile Devices

Xing Liu, *University of Minnesota, Twin Cities;* Christina Vlachou, *Hewlett Packard Labs;* Feng Qian and Chendong Wang, *University of Minnesota, Twin Cities;* Kyu-Han Kim, *Hewlett Packard Labs*

https://www.usenix.org/conference/atc20/presentation/liu-xing

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the 2020 USENIX Annual Technical Conference is sponsored by USENIX.

Firefly: Untethered Multi-user VR for Commodity Mobile Devices

Xing Liu *Christina Vlachou*[*] *Feng Qian* *Chendong Wang* *Kyu-Han Kim*[*]
University of Minnesota, Twin Cities [*]*Hewlett Packard Labs*

Abstract

Firefly is an untethered multi-user virtual reality (VR) system for commodity mobile devices. It supports more than 10 users to simultaneously enjoy high-quality VR content using a single commodity server, a single WiFi access point, and commercial off-the-shelf (COTS) mobile devices. Firefly employs a series of techniques including offline content preparation, viewport-adaptive streaming with motion prediction, adaptive content quality control among users, to name a few, to ensure good image quality, low motion-to-photon delay, a high frame rate at 60 FPS, scalability with respect to the number of users, and fairness among users. We have implemented Firefly in 17,400 lines of code. We use our prototype to demonstrate, for the first time, the feasibility of supporting 15 mobile VR users at 60 FPS using COTS smartphones and a single AP/server.

1 Introduction

Virtual Reality (VR) has registered numerous applications. In this paper, we focus on *multi-user VR* where multiple users jointly participate in exploring a VR scene. This enables many applications that single-user VR cannot support such as team training, social VR, group therapy, collaborative product design, and multi-user gaming.

We envision the following use case with more than 10 collocated users in a VR room. To start multi-user VR, each user simply launches the app on her smartphone and plugs the phone into a VR headset (*e.g.,* a $50 Samsung Gear VR [18] or even a $10 Google Cardboard [9] with a $6 VR controller [5]). These mobile devices fetch the VR content from an off-the-shelf server based on the users' real-time motion. The devices and the server communicate wirelessly over a single WiFi access point (AP). The users can enjoy the high-quality VR content as if it is rendered by a desktop PC with a powerful GPU. Meanwhile, each user can see and possibly interact with other users in the virtual world.

This paper aims at realizing the above ambitious use case. We design and implement Firefly, a novel multi-user VR system for mobile devices. The goals of Firefly are the following. First, Firefly works with affordable, commercial off-the-shelf (COTS) mobile devices, server, and AP. This helps reduce the deployment cost and facilitate the "bring-your-own-device" (BYOD) policies that many enterprises adopt today [6]. Second, Firefly employs untethered, wireless VR to overcome the inconvenience and trip hazards incurred by wired cables [19]. This is important for multi-user VR where multiple users' cables may easily get intertwined. Third, Firefly offers high content quality, low "motion-to-photon" (M2P) latency, and a high frame rate. An M2P higher than 16ms can cause nausea to VR users [11]. We target Quad HD (1440p) resolution, 60 frames per second (FPS) that can provide a good experience even for fast-paced VR gaming – the most demanding VR task [10]. Fourth, Firefly aims at supporting ∼15 users who can form a sizeable group of, for example, co-workers, students, or patients. To our knowledge, no existing system can achieve this using a single commodity server and WiFi AP. Recent work on multi-user VR only demonstrated 4 concurrent emulated users [47]. Fifth, Firefly allows complex VR scenes with both background and dynamic foreground objects, such as other users' avatars that users can interact with.

The above goals pose multiple challenges. The CPU/GPU power of a smartphone is at least one order of magnitude lower than its desktop counterpart [57], not to mention the energy/heat constraints; the heterogeneity of their computational capabilities should also to be taken into consideration; the bandwidth offered by a single AP is limited for multiple users; another key challenge is multi-user scalability, which calls for strategic decisions of splitting the client-server workload, as well as scalable approaches for rendering and distributing the content. To address the above challenges, Firefly makes a series of judicious design decisions as follows.

• Firefly performs one-time, offline content preparation by enumerating, pre-rendering, encoding, and storing the views at all positions reachable in a virtual scene [27]. At runtime, given a user's position and viewing direction, the server directly retrieves the stored high-quality content and delivers it to the user. This completely eliminates the online rendering overhead. Prior work [27] applies offline rendering to a single mobile device for local VR scenes, while Firefly further extends this concept to networked multi-user VR where offline rendering is found to be an indispensable mechanism ensuring scalability (§3.1).

• To reduce the network bandwidth consumption, Firefly takes a viewport-adaptive approach: each user only requests for the content that the user is about to perceive based on motion prediction. We conduct a thorough analysis of 25 human users' motion traces collected from an IRB-approved user trial. The results shed light on developing a lightweight yet effective motion prediction approach for Firefly. In the literature, several studies [24, 33, 39] have examined 360° video viewers'

viewing patterns that only involve rotational movement (yaw and pitch). Our study instead investigates generic VR users' motion that consists of both the rotational and translational viewport movement as well as their interplay (§3.2).

• Firefly supports *Adaptive Quality Control (AQC)*, which determines the content quality of each user based on the total network bandwidth, the bandwidth available to each user, and the amount of to-be-delivered content. AQC essentially extends traditional video bitrate adaptation [40, 41, 51, 66]: from handling a single client to multiple clients, from dealing with regular videos to immersive VR content, and from being invoked at the second level to the millisecond level to adapt to users' motion. These differences require AQC to be effective, lightweight, fair, and scalable as reflected in our design (§3.4).

• Firefly handles dynamic foreground objects in a scalable and adaptive manner. Specifically, objects' 3D models are distributed to the clients offline. They are then rendered locally by the client. This eliminates the uncertainty caused by the network as well as the potential resource competition from other users compared to a server-side approach. To prevent too many objects appearing in the viewport from slowing down client-side rendering, Firefly supports adaptively reducing the objects' fidelity to maintain a high FPS (§3.6).

Additionally, Firefly has integrated several system-level optimizations, such as motion prediction error toleration (§3.3), client-side hierarchical cache (§3.5), and AP-assisted bandwidth estimation (§4). Our implementation on commodity Android/Linux platforms involves 17,400 lines of code. We conduct extensive evaluations using commercial VR scenes, real users' motion traces, and off-the-shelf smartphones/AP/server. We highlight the evaluation results as follows (§5).

• Firefly achieves very low motion-to-photon delay ($\leq$15ms for 99% of the frames), low stall duration (around 1 second per minute), a frame rate at 60 FPS, and fairness among the users when supporting 15 concurrent users with a single server and a single 802.11ac AP (§5.2).

• Firefly is adaptive to users dynamically joining and leaving the system as well as network bandwidth changes (§5.4,§5.5).

• Firefly significantly outperforms existing systems. We extend Furion [44], a state-of-the-art single-user VR system over WiFi, to support multi-user VR. Due to its more efficient content fetching strategy, Firefly exhibits 18% higher median FPS, 6.9$\times$ lower stall duration, and much higher content quality, compared to multi-user Furion (§5.2). We also use our 15-user dataset to evaluate MUVR [47], a very recently proposed multi-user mobile VR framework. Through simulation, we find that for 27% of the time, the MUVR server still needs to perform online rendering for more than 5 devices. This makes MUVR not scalable to many users (§5.6).

• Firefly incurs acceptable CPU, GPU, and memory usage. When tested on 5 modern smartphones, after 25-minute VR sessions, the battery life percentage drops by 4% to 8%, and the devices' temperature reaches no higher than 50°C (§5.7).

Firefly is to our knowledge the first system that can scale untethered multi-user mobile VR. We make multi-fold contributions in this work: (1) the design of Firefly, (2) the study of real VR users' motion, and (3) our prototype implementation that demonstrates the support of 15 VR users at 60 FPS using COTS smartphones and a single AP/server. With emerging wireless technologies (*e.g.,* 802.11ax and 5G), we believe that Firefly has the potential to scale up to even more users.

2 Motivation and Overview

Firefly enables multiple users (10+) to simultaneously enjoy high-quality VR at 60 FPS using commodity smartphones, a single off-the-shelf server, and a single WiFi access point. We consider three high-level architectural design options.

A Serverless Design does not involve a server, so all the VR content is stored on users' mobile devices, which also perform full-fledged rendering. Most of today's commercial 3D games and VR mobile apps use this approach. However, previous studies [27, 44] indicate that today's commodity mobile devices are far from being powerful enough to perform heavy-duty real-time rendering for high-quality VR. Other concerns include excessive energy consumption and heat dissipation.

Server Performing Online Rendering. This design option offloads the rendering task to an (edge) server, which performs real-time rendering of the VR scene for all users based on their positions and viewports. The rendered scenes are then distributed to the users wirelessly as encoded video frames. This approach has been adopted by a prior single-user, cloud-assisted VR system [44]. It drastically reduces the client-side overhead, but in the multi-user scenario, the rendering and video encoding workload becomes too high for a single server to handle. To illustrate this, we perform an H.264 encoding experiment on a high-end workstation equipped with an Nvidia GTX 1080 GPU. The achievable encoding performance is 92 FPS, 199 FPS, and 342 FPS for 4K, 2K, and 1080p resolutions, respectively. This clearly cannot support 10+ users, each requiring a frame rate of higher than 60 FPS.

Server Performing One-time, Exhaustive Offline Rendering. The server exhaustively enumerates all possible views at all positions, renders them at a high quality, encodes them into video frames, and saves the frames in the storage [27]. At runtime, the server simply retrieves and transmits the pre-encoded frames based on each user's position and viewport. In this way, the rendering/encoding overhead at runtime is completely eliminated, so the server can easily scale to tens or even hundreds of simultaneous users. These benefits come at the cost of high storage usage, which is largely not an issue given the cheap storage today.

System Architecture. Firefly employs the third approach given its good runtime performance and superior scalability. Figure 1 plots the overall architecture. As shown, Firefly consists of a content server and multiple commodity mobile

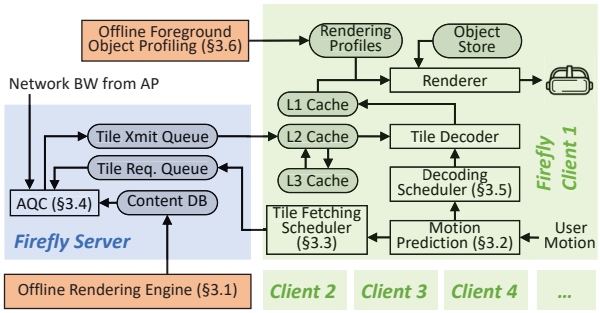

Figure 1: The Firefly system architecture.

devices. They are wirelessly connected through a WiFi access point (AP). This setup can be easily realized in enterprise or home environments at a very low cost. Note that prior work [27] applies offline rendering to a single mobile device for local VR scenes, while Firefly further extends this concept to networked multi-user VR where offline rendering is found to be an indispensable mechanism ensuring the scalability.

The server consists of a content database that stores rendered/encoded content indexed by a user's position and viewing direction. The database is built by the Offline Rendering Engine that performs the aforementioned exhaustive content generation (§3.1). Another critical component is the AQC module that is introduced to scale the system and to handle the wireless bandwidth fluctuation. It determines in real-time the content quality for each user. Designing AQC is challenging due to multiple requirements including boosting users' QoE, maintaining good performance, ensuring scalability, and achieving fairness. We detail its design in §3.4.

On the client side, there are two high-level design choices on the content fetching strategy for background frames. First, the client can prefetch all surrounding frames at every new virtual position [44]. However, this technique may consume high bandwidth with a considerable amount of wasted traffic (*i.e.,* the fetched content is not viewed by the user, see our evaluation in §5.2), making it infeasible for multi-user VR. Second, to reduce the bandwidth footprint, the client can use its historical motion trajectory to predict the future viewport and to prefetch only the portions that will likely be consumed in the near future. Firefly is the first to incorporate this viewport-adaptative approach into generic VR using robust motion prediction (§3.2,§3.3). The client also efficiently manages its local cache (§3.5) and handles foreground dynamic objects in an adaptive and scalable manner (§3.6).

3 System Design

3.1 Offline Rendering Engine

The offline rendering engine produces the content database. The whole VR world is discretized into grids. At each grid position that the user can reach, the rendering engine renders a *mega frame* that captures the 360° panoramic view [28] that the user can possibly perceive at a high quality. Firefly uses Equirectangular projection [7] to generate the panoramic representation, but other projection algorithms [8, 14, 67] can also be applied. As shown in Figure 2, besides the color frame (top), a mega frame also includes a panoramic depth map (bottom) where the brightness of each pixel indicates its distance from the user. The depth map will be used to ensure the correct occlusion when overlaying foreground objects such as avatars of other users onto the scene (§3.6).

We next apply the *tiling* technique [38, 53] by dividing each mega frame into *mega tiles*. Each tile is independently encoded and can be separately transmitted and decoded. The rationale is that, since the user only sees a portion of the whole panoramic scene at a given time, there is oftentimes no need to fetch the entire mega frame. The mega tiles thus allow users to (pre)fetch the content more adaptively at a finer granularity, to reduce the network bandwidth consumption. Note that although viewport-adaptive tiling has been used in 360° video streaming, applying this concept to generic VR (in particular, multi-user VR) is new. Tiling requires the user to predict its viewport, *i.e.,* to determine which tiles to fetch based on the observed viewport trajectory (both translational and rotational), as to be detailed in §3.2 and §3.3.

A decision we need to make is to determine the number of tiles and their layout. While having more tiles provides more bandwidth saving opportunities, in the meantime it increases the decoding overhead and makes compression less efficient. After carefully studying the above tradeoffs using real users' viewport trajectory data (§3.2), we decide to vertically segment each mega frame into four mega tiles as shown in Figure 2. We choose vertical segmentation because according to our data collected from 25 users, users tend to keep their sight vertically centered (*i.e.,* looking at the equator) while moving the viewport horizontally. This makes horizontal segmentation at the equator (0° latitude) inefficient because the vertically centered viewport will always overlap with at least two tiles, *i.e.,* one above and the other below the equator.

As described above, at each position, the offline rendering engine generates four tiles capturing the panoramic view and depth. Each tile is then independently encoded into video frames with multiple quality levels. The rendered and encoded tiles are stored in the content database, indexed by the user's grid (translational) position, the tile ID (rotational position, 1 to 4), and the quality level.

3.2 VR Viewport Movement: Characterization and Prediction

Users' motion makes VR immersive and interactive. In the literature, many studies have investigated users' head *rotational* movement when watching 360° videos [24, 33, 39]. Generic VR differs from 360° videos in that it further involves *translational* movement. To our knowledge, no prior study has comprehensively investigated VR users' motion patterns and their predictability, which are our focus here.

Collecting Viewport Movement Data from Real Users.

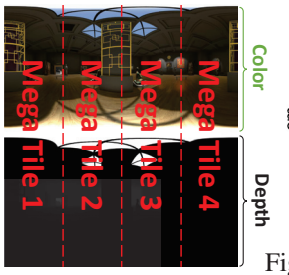

Figure 2: Mega frame.

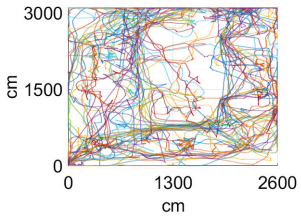

Figure 3: Users' translational trajectories (the Office scene).

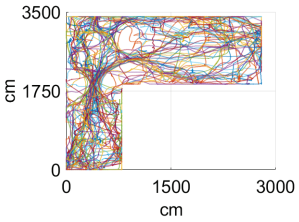

Figure 4: Users' translational trajectories (the Museum scene).

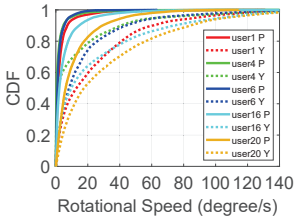

Figure 5: Five users' rotational speed (P=Pitch, Y=Yaw).

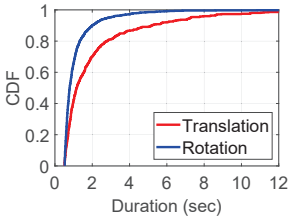

Figure 6: SP duration per pause.

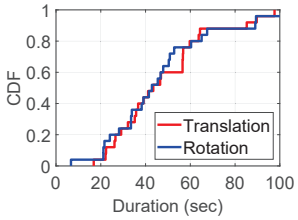

Figure 7: Total SP per user.

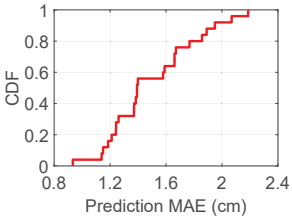

Figure 8: Trans. prediction MAE.

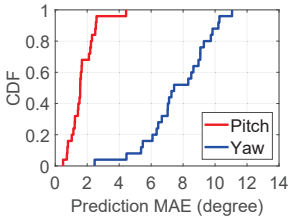

Figure 9: Rot. prediction MAE.

We conduct an IRB-approved user study involving 25 voluntary participants recruited from a large university. Among the 25 users, 9 are female. The users are from 8 departments as undergraduate (16), master (4), and Ph.D. students (5). During the study, each subject wears an Oculus Rift headset [15] connected to a high-end PC. The subject can freely make rotational movement by moving her head as well as perform translational movement using the handheld controller.

We obtain two large VR scenes from the Unity store: Office [16] (30m×26m) and Museum [13] (35m×30m, L-shape). We then develop a custom VR system that loads each scene for the users to explore. Our system logs from each user the precise viewport trajectory. We let each subject explore each scene in a random order for 5 minutes, with an arbitrarily long break allowed between the two sessions.

Motion Trace Characterization. We now characterize the unique dataset above to reveal VR users' motion dynamics and to provide insights for Firefly's design. To begin with, Figures 3 and 4 plot the translational movement trajectories of all users, represented by different colors, for the two VR scenes. As shown, in most locations, the users' trajectories are highly heterogeneous. This finding suggests that the server should not use broadcast or multicast, simply because users typically see different content at a given time.

Fast motion may cause difficulties for viewport prediction. We thus quantify the users' motion speed. The translational movement speed is fixed at 1m/s (set based on reported experiences from another user study) when the user presses the controller button. Figure 5 plots the distributions of rotational movement speed, calculated by sliding a 500ms window over the trajectory, across all window positions for five randomly selected users. As shown, the users exhibit different speeds, whose medians range from 1.3°/s to 18.6°/s for yaw and from 0.5°/s to 7.0°/s for pitch. The median speed across all 25 users is 10.2°/s and 2.4°/s for yaw and pitch, respectively. Interestingly, such speeds match those for typical 360° users [53], implying that translational movement does not necessarily slow down the rotational movement.

Another challenging scenario is users' sudden movement after a stationary period. How often do stationary periods (SPs) occur? Figure 6 plots the distributions of SP duration per pause, which by our definition has to last at least 500ms. Figure 7 plots the total SP duration per user. As shown, an SP is typically short: 69% of translational SPs and 89% of rotational SPs are shorter than 2 seconds. However, Figure 7 indicates that they occur frequently: within a 5-min VR session, a typical user spends 43 seconds (median) being stationary. Such frequent SPs lead to bursty, non-continuous movement patterns that pose difficulties for viewport prediction. To deal with SPs, we design mechanisms such as conservative tile scheduling (§3.3) and bandwidth reservation (§3.4). We also find that translational and rotational SPs are not correlated, *i.e.,* a user is typically looking at a fixed direction while moving, or looking around while standing still. This motivates us to separate the translational and rotational dimensions when performing viewport prediction (see below).

Viewport (Motion) Prediction is required by the tiling scheme (§3.1). We make two decisions regarding Firefly's viewport prediction scheme. First, we decide to run it distributively on client devices to make the server scalable. Second, given the above measurement results, we predict each dimension separately (yaw/pitch for rotational movement and X/Y/Z for translational movement), and then combine them into the final predicted view. We find that this strategy greatly reduces the computational complexity while achieving a decent accuracy – a desirable tradeoff we want to strike. Regarding the actual algorithm, we continuously train a linear regression (LR) model using the motion trajectory observed within a history window of H milliseconds; we then use this model to predict the future trajectory within a prediction window

of P milliseconds before discarding the model. The simple LR model is found to be very lightweight yet effective for 360° videos [53]; here we investigate its effectiveness for generic VR motion prediction. Further improvement using more powerful machine learning tools is our on-going work. Ideally, P should be set to the duration of the entire tile processing pipeline (form request being sent to tiles being decoded) plus some safety margin. Guided by this, we set P to 150ms based on empirical profiling. We set H to 50ms based on cross-validating different values of H, which is found to not qualitatively impact the prediction accuracy. Note that when integrated with Firefly, the prediction is performed in an online manner: at runtime, Firefly continuously (1) trains a linear regression model based on the motion trajectory observed within a window (H), (2) uses this model to predict the viewport, and (3) discards this model immediately.

Figure 8 and 9 plot the prediction results for translational and rotational movement, respectively, across all users (H=50ms, P=150ms, the Office scene), with the SPs excluded. The accuracy metric is the mean absolute error (MAE, in distance or degree). The overall accuracy is high: the median MAE is around 1.4 cm for translational movement, and 1.6°/7.4° for vertical (pitch) / horizontal (yaw) rotational movement. The results for the Museum scene are similar. We discuss how Firefly further tolerates prediction errors in §3.3.

3.3 Client-side Tile Fetching Scheduling

The client needs to judiciously decide which (mega) tiles to fetch and in which order. Recall that the client continuously predicts the viewport trajectory within a prediction window (§3.2). The trajectory is a time series of 6-tuples $\{t, x, y, z, pitch, yaw\}$ where t is the (future) timestamp; x, y, and z are the grid position (translational movement); $pitch$, and yaw are the viewing direction (rotational movement). The timestamp difference between two consecutive tuples is $1/F$, where F is the frame rate. In other words, each tuple corresponds to the predicted viewport of a future frame. The client then translates yaw and $pitch$ of each tuple into a list of tiles according to the projection algorithm (*e.g.,* Equirectangular).

The client now has a preliminary list of tiles to be fetched. It next prunes the list using two rules. First, if a tile is already in a client-side cache (§3.5), it will be removed from the list. Second, if a tile appears multiple times in the list, only the earliest appearance (with the smallest t) will be kept. This pruned list where the tiles are ordered by their t values will then be sent to the server. To adapt to users' motion, the above scheduling process is performed continuously on a per-frame basis. The server therefore sees a stream of mega tile lists for each user. We describe how the server processes it in §3.4.

Tolerating Viewport Prediction Errors. Due to users' randomness, viewport prediction errors are inevitable. Firefly employs three mechanisms to tolerate them. First, it uses large tiles (90°×180°) that can absorb rotational prediction errors, as a tile needs to be fetched as long as the predicted viewport has any overlap with it. Second, to further tolerate rotational prediction errors, we virtually enlarge the field-of-view by p% in each direction when calculating the to-be-fetched tiles. p is configured to 10% given the rotational prediction MAE shown in Figure 9. Third, recall from §3.2 that sudden translational movement after a stationary period (SP) is difficult to predict. To address this issue, when the user is stationary, we add the tiles (corresponding to the current viewing direction) of all four neighboring grids to the predicted tile list. In this way, no matter which direction the user moves towards, the corresponding tiles are always in the to-be-fetched list.

3.4 Adaptive Quality Control (AQC)

AQC takes as input the lists of tiles requested by the users, and outputs each user's appropriate quality level. It runs on the server that has the global knowledge of all users. An ideal AQC algorithm has the following features. (1) For each user, AQC will maximize the quality level while minimizing the stall (rebuffering); meanwhile, the number of quality switches should be minimized to provide a smooth user experience. (2) The selected quality levels should be fair across all the users; in other words, the quality levels should be largely proportional to the users' wireless channel capacities. (3) AQC needs to execute in a fast-paced manner (ideally at the per-frame granularity for each user) to adapt to users' motion. (4) AQC should scale well for multiple users.

At a first glance, AQC is similar to a video bitrate adaptation algorithm where a plethora of studies have been conducted [40,41,51,66]. However, AQC in Firefly is much more challenging. In particular, requirements (2), (3), and (4) do not appear in typical bitrate adaptation algorithms running on a single client for regular video-on-demand services.

In our initial design, we attempt to establish a principled optimization framework that maximizes a QoE (Quality of Experience) utility function. However, we find that this approach is computationally infeasible on a per-tile basis, as the solution space expands exponentially as the number of users increases. To this end, we develop a lightweight, heuristic-based algorithm that produces empirically good quality selection decisions. Our design considers all four requirements mentioned above. It runs efficiently on commodity servers, achieving frame-level scheduling for 10+ users.

AQC Algorithm. We now walk through the detailed logic of the algorithm listed in Figure 10. It uses the available bandwidth obtained from the wireless AP and the recently received to-be-fetched tiles (§3.3) to adjust the quality level (`Q[i]`) for each user `i`. In each invocation, AQC gets the total available downlink bandwidth across all users (Line 01), as well as each individual user's available downlink bandwidth from the AP (Line 03). They represent the global and local network bandwidth constraints respectively (see §4 for their details). λ (empirically set to 90%) adds a safety margin for

```
n: total number of users
T: total available bandwidth across all users
Q: users' current quality levels (input & output)
Tiles: users' to-be-fetched tile lists (input)
Q': local copy of Q
B: individual user's available bandwidth
λ: bandwidth usage safety margin
RESERVE: reserved bandwidth for each user
```

```
01 T = get_total_bw_from_AP() * λ
02 Q'[1..n] = Q[1..n]
03 B[1..n] = get_individual_bw_from_AP([1..n]) * λ
04 foreach user i:
05     while (bw_util(Tiles[i],Q'[i])≥B[i] and Q'[i] is not lowest):
06         Q'[i] = Q'[i] - 1
07     T = T - min(B[i], max(RESERVE, bw_util(Tiles[i], Q'[i])))
08 if (T < 0):
09     lru_decrease(Q'[1..n]) until (T≥0 or Q'[1..n] are lowest)
10 else:
11     lru_increase(Q'[1..n]) until (T≈0 or Q'[1..n] are highest)
12 Q[1..n] = Q'[1..n]
```

Figure 10: The multi-user AQC algorithm.

tolerating the bandwidth fluctuation. Lines 05–06 deal with the local bandwidth constraint. For a given user `i`'s tiles to be fetched (`Tiles[i]`), as long as their bandwidth utilization (calculated by `bw_util()`) exceeds the available bandwidth `B[i]`, the quality is lowered to avoid stalls. Line 07 then subtracts the user's used/reserved bandwidth from the global bandwidth budget `T`. An important design decision we make is to reserve a certain amount of bandwidth (`RESERVE`) for each user to handle the user's sudden movement (§3.2) that may incur unexpected bandwidth utilization. The reserved bandwidth for each user is set to $\eta T/n$ where T is the AP's total bandwidth, n is the number of users, and η is a tunable parameter. A large η reserves more bandwidth, which can help increase the resilience to users' bursty movement at the cost of a lower flexible (*i.e.,* non-reserved) bandwidth of other users. We empirically choose η=0.75 that yields a satisfactory tradeoff between the two above factors.

We next consider how to estimate the tiles' bandwidth requirement, *i.e.,* realizing `bw_util()` in Line 05. Recall from §3.3 that each tile has its display deadline. Let the total size (in bytes) of the tiles at quality level q with a deadline at or before t_i be $S_{i,q}$. Let t_0 be the current time, t_c be the estimated decoding time, and t_s be the server-side queuing delay. t_0 and t_c are reported by the user and t_s is estimated by the server. In order to not miss the deadline t_i, the required bandwidth should be at least $b(t_i) = S_{i,q}/\max\{0,(t_i - t_0 - t_c - t_s)\}$ (it can be ∞ when a stall occurs). Then the overall required bandwidth is conservatively estimated as $\max_{t_i}\{b(t_i)\}$.

Lines 08 to 11 deal with the global bandwidth constraint. If the global bandwidth budget `T` is depleted (Line 08), then we reduce the users' quality levels (Line 09); otherwise we try to increase them (Line 11). To facilitate fairness and make the quality switch smooth, the decrease/increase of the quality levels is performed in a "least recently used (LRU)" manner, one user at a time, *i.e.,* the user whose quality level was least recently changed is selected. The quality level increase is subject to the local bandwidth constraint.

Since users' requests arrive asynchronously, AQC needs to be invoked to update `Q[1..n]` whenever a new request arrives. Then the tile transmission thread will retrieve the tiles from the content database and put them into the tile transmission queues. If the requested tiles for a user change, or if AQC produces a different schedule in a future invocation for this user, the not-yet-transmitted tiles in the user's queue will be updated. Thanks to AQC's lightweight nature, users' motion and network bandwidth fluctuation will be immediately reflected in the tiles' quality levels, making Firefly robust.

3.5 Client-Side Hierarchical Cache

When a user receives mega tiles from the server, the tiles will be cached, decoded, and rendered. Since tile decoding takes non-negligible time, it needs to be performed in advance. Firefly, a *decoding scheduler* determines which tiles to decode. Its logic is similar to the tile fetching scheduler (§3.3), by using the viewport prediction results. Predicted tiles with a closer display deadline take a higher decoding priority.

To handle large VR scenes, Firefly needs to fetch and decode a large number of tiles. Firefly thus employs a 3-layer hierarchical tile cache. Borrowing the CPU cache terminologies, we name the three layers L1, L2, and L3. Residing in the GPU memory, The L1 cache stores *decoded* mega tiles that can be immediately rendered by the GPU. It is the fastest cache, but its capacity is the smallest (hundreds of tiles) due to the large size of decoded tiles and limited GPU memory. The L2 cache stores *encoded* tiles in the main memory with a capacity of thousands of tiles. The L3 cache dumps encoded tiles in the persistent storage; it has the largest size but is the slowest. When a tile arrives, it is first stored in L2 cache; if L2 is full, some old tiles in L2 may be swapped to L3 in an LRU manner. The L2-to-L3 swap involving writing to flash drive, and is thus performed in a batched manner for good write performance. Swapping back from L3 to L2 is triggered by the decoding schedulers' decisions. This typically occurs when a user visits a previously explored grid position.

3.6 Handling Dynamic Foreground Objects

A VR scene may consist of a background view as well as one or more foreground objects. The background view at a specific virtual location is static. Due to its large area and complexity, its rendering typically dominates the workload for preparing the scene. In contrast, foreground objects are more dynamic and less complex than a background scene. Their examples include moving objects (*e.g.,* other users' avatars) and interactive objects (*e.g.,* a virtual control panel). Despite being less complex than the background view, due to their dynamic and interactive nature, failure to render foreground objects in time may also cause considerable QoE degradation.

Firefly employs two mechanisms to handle foreground objects. First, objects' 3D models (polygons, textures, *etc.*) are distributed to the clients *offline*. This reduces the network bandwidth consumption and eliminates the server's rendering workload at runtime. In a typical VR scene, the objects' 3D

Quality	High	Medium	Low
# Polygons, Size (MB)	30016, 3.30	14566, 1.30	7283, 0.68

Table 1: Three quality levels of the avatar object.

Client Device	High	Medium	Low
Samsung Galaxy S8 (SGS8)	✗,✗,✗	✓,✓,✗	✓,✓,✓
Samsung Galaxy S10 (SGS10)	✓,✓,✓	✓,✓,✓	✓,✓,✓
Samsung Galaxy Note 8 (SGN8)	✓,✗,✗	✓,✓,✗	✓,✓,✓
Motorola Moto Z3 (Z3)	✓,✗,✗	✓,✓,✓	✓,✓,✓

Table 2: Rendering profiles for different phones: whether 60+ FPS can be achieved with 3,6,9 concurrent objects in different qualities.

models are not large (*e.g.,* tens of MBs in total) so they can be bundled with the app installation package or be fetched when the app launches for the first time.

Second, foreground objects are rendered locally by the client. This eliminates the uncertainty caused by the network as well as the potential resource competition from other users compared to a server-side approach. A challenge here is that the number of objects appearing in the viewport may change dynamically. If there are too many objects, the local rendering may still become the bottleneck. For example, in multi-user social VR, a user "sees" other users as 3D avatars; depending on the users' position, more than 10 avatars may appear in the viewport, incurring high rendering overhead. To address this challenge, Firefly supports trading off the rendering quality for a high frame rate. Specifically, the client creates low-quality versions for each object type by downsampling its polygon meshes. Table 1 shows an example of an avatar object originally with 30K polygons. Firefly downsamples them (using Blender [4]) to the medium and low quality with 14.6K and 7.3K polygons respectively. This downsampling process is an offline, one-time effort. Then at runtime, depending on the number of objects to be rendered, their qualities are dynamically determined to facilitate 60 FPS. Downsampling may also use more sophisticated polygon simplification techniques such as bounded-error polygon simplification [42], progressive encoding [45], or adaptive display elision based on the size of the object and its position in the scene [34].

To properly determine the objects' qualities, each client creates a *rendering profile* offline. Let us first assume that there is only one object type (*e.g.,* the avatar). As exemplified in Table 2, for each quality level, the profile maps the number of concurrent objects (3, 6, 9 are shown) to whether 60+ FPS can be achieved. Note that we assign the same quality to all objects to simplify the quality selection. The profile is created by the client through automated tests. During a test, the client is also performing tile decoding/rendering to mimic the workload of generating the background view. Then at runtime, the client can directly consult its profile to determine the objects' quality level. For example, when there are 6 objects, SGS8 should use the medium quality (Table 1) to achieve 60 FPS. When there are multiple types of objects, it may be infeasible to exhaustively enumerate their combinations. In this case, we can apply simple machine learning to predict the rendering performance, using features such as the number of objects, the total number of polygons, *etc.* We leave this as future work.

4 System Implementation

Client and Server. We have integrated the components in §3 into the holistic Firefly system that works on commodity Android/Linux OSes. The client is implemented using Android SDK with a total line of code (LoC) of 14,900. Tile decoding is realized using the low-level Android MediaCodec API [3]. We leverage multiple concurrent hardware decoders, whose optimal number depends on the device, to boost the decoding performance. We use OpenGL ES to perform tile projection/rendering, and use the OpenGL FBO (Framebuffer Object) to realize the L1 decoded cache (§3.5). We have successfully tested Firefly on four mobile devices: SGS8, SGS10, Moto Z3, and SGN 8 (full names in Table 2). These devices can be readily plugged into affordable VR headsets. The rotational and translational motion is provided by the on-device motion sensors and the VR headset controller, respectively. The server is implemented on Ubuntu 16.04 with about 1,000 LoC. The clients and the server communicate over TCP.

WiFi AP. The clients and server are wirelessly connected by a commodity WiFi AP. Since the server is only one wireless hop away from the users, AQC can directly obtain accurate global and per-user available bandwidth from the AP (Line 01 and 03 in Figure 10). This avoids the error-prone bandwidth estimation process widely used in Internet video streaming. To obtain the AP-wide overall bandwidth, we modify the AP's firmware to collect statistics on the maximum PHY rates of the clients, the wireless bandwidth used (20–160MHz in 5GHz Wi-Fi bands), and the busy channel time from hardware registers. To estimate each user's available bandwidth, we also collect statistics on the PHY rate and the frame error rate. The available bandwidth for a client i is estimated as $\Phi_i(1-\varepsilon_i)(1-U)O^{TCP}/N$, where Φ_i is its PHY rate, ε_i is the error rate, U is the channel busy airtime, N is the number of clients taking into account that the airtime will be shared fairly among clients, O^{TCP} is the TCP overhead estimated offline through bandwidth saturation experiments. Similar statistics are available on other APs via interfaces such as WebUI.

The Offline Rendering Engine (§3.1) consists of a rendering engine (developed in C# using Unity) and a mega tile encoder (developed in Python) with a total LoC of 1,500. We use H.264 encoding supported by all mainstream mobile devices.

5 Evaluation

5.1 Evaluation Setup

Content Preparation. We use two commercial VR scenes purchased from the Unity store: Office [16] (30m×26m) and Museum [13] (35m×30m, L-shape). The offline rendering engine (§3.1) discretizes both scenes into 5cm×5cm grids, which are fine-grained enough to provide a smooth translational movement experience. The offline engine renders each panoramic frame in 1440p (Quad HD, 2560×1440) resolu-

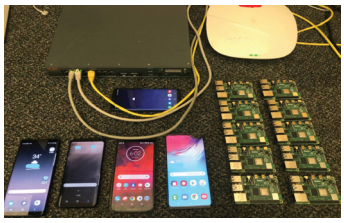

Figure 11: Our equipment: phones, Raspberry Pis, and WiFi AP.

tion, and encodes each mega tile into four quality levels, using the following CRF (Constant Rate Factor) values: 19, 23, 27, 31. A higher CRF corresponds to a lower quality and a lower encoded bitrate. The CRF values are selected by following prior recommendations [17, 20] where the encoded bitrate ratio between two neighboring quality levels is approximately 1:1.5. The content database size is 137 GB and 99 GB for Office and Museum, respectively. When exploring each scene, a user can see other users as avatars, which are rendered by the client as foreground objects. The statistics of the avatar's three quality levels are listed in Table 1.

Hardware and Software. As shown in Figure 11, we use 15 client devices. 5 of them are COTS smartphones with different computational capabilities: SGS8×2 (released in 2017), SGN8 (2017), Z3 (2018), and SGS10 (2019). They all run unmodified Android 9.0. For the remaining clients, we use 10 Raspberry Pi 4 (model B) to emulate them, each having a quad-core ARM Cortex-A72 CPU @ 1.5GHz and 2GB memory. The Pis run Raspbian OS (Debian v10 with Linux kernel 4.19). We run full-fledged Firefly on the 5 smartphones. For the Pis, we create an emulated version of Firefly by replacing the decoding and rendering components with their emulated counterparts. The decoding/rendering latency is properly emulated using the numbers profiled from the 5 smartphones. All other components such as AQC, viewport prediction, tile fetching scheduler, decoding scheduler, L2/L3 caching are identical to those running on a real Firefly client. The server is a desktop PC with an octa-core CPU @ 3.6GHz, 16 GB memory, 1TB disk, and Ubuntu 16.04. The server does not have a dedicated GPU. Clients and server are connected by an Aruba AP running 802.11ac on 80MHz bandwidth.

Physical Environment. The experiments are conducted in a typical office room (7.9m×7.3m) where all the devices, the server, and the AP are located. We distribute the devices at random locations. We find that their locations have a small impact on network performance. For a single device, placing it at the spot nearest to the AP and the spot furthest from the AP yields a throughput difference no more than 11%.

Experimental Approach. To ensure reproducibility, our high-level experimental approach is to replay real users' motion traces collected in §3.2. Recall that we have 25 user traces and 15 devices. In each run, we randomly pick 15 users and assign them in a random order to the devices. Each device then replays the corresponding user's motion trace by feeding the sensor stream to Firefly with precise timing. By default, each experiment consists of 5 back-to-back runs with different user-to-device assignments. We set the users' field-of-view (FoV) to a typical value of 100°×90° [53]. Unless otherwise mentioned, the presented results are based on the Office scene as the results for the Museum scene are qualitatively similar.

5.2 Overall Performance Comparison

We first evaluate the overall performance of Firefly, with the following metrics. (1) **Missed Frame Count (MFC).** In our client implementation, a high-precision rendering timer is triggered every 15ms (or 66.67 Hz). If a frame is not ready at the current timer event, it needs to wait for the next timer event, *i.e.,* after 15ms. In this case the client reports one MFC. MFC is highly correlated with the motion-to-photon delay [69], the time needed for a user's motion to be reflected on the display. When MFC=0 (the ideal case), the motion-to-photon delay is minimized to no longer than 15ms, *i.e.,* the motion is reflected in the immediate next frame. When MFC>0, a stall occurs. (2) **Average frame rate** is measured by sliding a 1-second window over a VR session and calculating the *average* FPS within each window. Our target FPS is 60. (3) **Stall duration** is the rebuffering time experienced by a user. We normalize it to seconds per minute for a VR session. This metric is correlated with the MFC. (4) **Content Quality.** Recall that a tile's quality is defined by the CRF value ∈{19,23,27,31} (§5.1). We then define the quality of a frame as the average quality of all tiles visible in the viewport. (5) **Inter-frame Quality Variation** is measured by sliding a 1-second window over a VR session and calculating the standard deviation of all frames' quality values (defined above) within each window. Since frequent quality switches degrade the QoE [66], a lower value of this metric is preferred. (6) **Intra-frame Quality Variation** of a frame is defined as the standard deviation of the quality values of all tiles appearing in a frame's viewport. Similar to the inter-frame quality variation, we prefer a lower intra-frame quality variation. Metrics (4), (5), and (6) are defined for the background view only. We evaluate the adaptation mechanism for foreground objects in §5.3.

Approaches to Compare. We compare three approaches: (1) full-fledged Firefly, (2) full Firefly with perfect prediction, and (3) the multi-user version of Furion [44]. Approach (2) represents an ideal scenario where users' viewport trajectories are known a priori. It helps us understand how much performance improvement we can further gain by having the perfect knowledge of users' motion. Regarding Approach (3), Furion is the state-of-the-art solution for single-user untethered VR. We create a multi-user version of Furion as follows. We use the full Firefly as the base (to handle multi-user), and then make (and only make) the following modifications according to Furion's design. First, we remove viewport prediction that Furion does not perform. Second, Furion does not use viewport-adaptation; the client instead always requests for all tiles belonging to all four neighboring grids; we thus modify

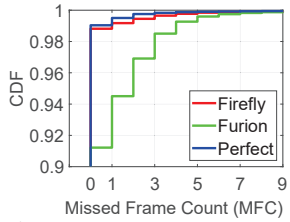

Figure 12: Missed frame count.

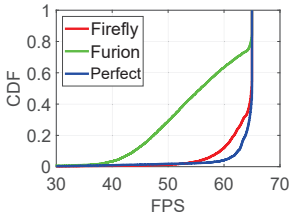

Figure 13: Average FPS.

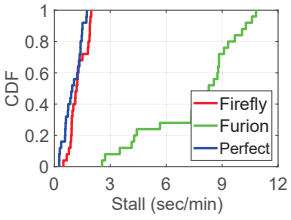

Figure 14: Stall duration.

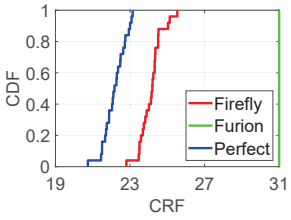

Figure 15: Content quality.

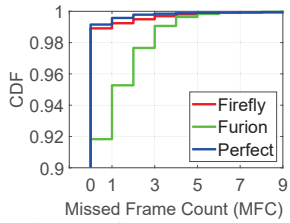

Figure 16: MFC (Museum).

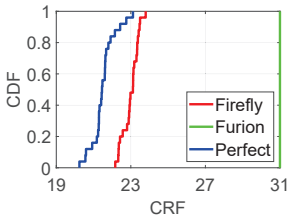

Figure 17: Quality (Museum).

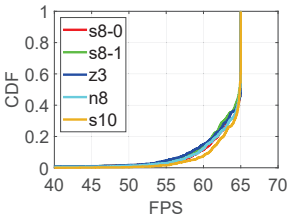

Figure 18: FPS fairness.

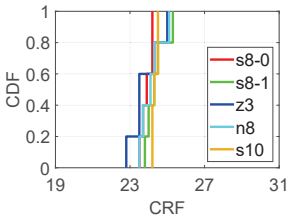

Figure 19: Quality fairness.

the tile scheduling module (§3.3) accordingly.

We next present the results for the Office scene. Figures 12, 13, 14, and 15 plot the distributions of the aforementioned four metrics: MFC (across all timer events), average FPS (across all 1-sec windows' measurements), stall (across all users' sessions), and average content quality (across all users' sessions). Thanks to its adaptiveness to available network/computation resources and its resilience to motion prediction inaccuracy, Firefly achieves overall good performance across all these metrics, which are the same or only slightly worse compared to Firefly with perfect prediction. Specifically, (1) 99% of the timer events (99% for perfect prediction) have MFC=0, *i.e.,* a motion-to-photon delay $\leq$15ms; (2) for 90%/99% of the 1-sec windows (95%/99% for perfect prediction), the average FPS is at least 60/50 FPS; (3) the median stall duration is only 1.2 sec/min (1.0 sec/min for perfect prediction); (4) the median content quality is around CRF 24.2 (CRF 22.2 for perfect prediction). In Figure 15, the slightly lower quality compared to that of perfect prediction is due to the additionally fetched tiles. The bandwidth consumed by these tiles is wasted because they are not viewed by the users due to viewport prediction errors.

The multi-user Furion exhibits much worse performance. This is because without prediction, it can only blindly fetch an excessive number of tiles without any prioritization. As a result, the bandwidth consumed of many tiles is wasted, leading to a much lower content quality; wasted tiles may also cause head-of-line blocking for useful tiles, causing stalls and a lower FPS. The results for the Museum scene are qualitatively similar, as exemplified in Figures 16 and 17, which plot the MFC and content quality results, respectively.

We also measure the inter/intra-frame quality variations, and find them to be low. For Firefly, the 25th, 50th, and 75th percentiles of the inter-frame quality variation (across all 1-sec windows' measurements) are 0, 0.2, and 0.3, respectively; the 90th percentile of the intra-frame quality variation is 0. Both metrics are very close to Firefly with perfect prediction. The low quality variations are attributed to AQC's quality selection mechanism. It (1) assigns the same quality to all the tiles in a viewport and (2) performs LRU-style quality changes that not only ensure fairness (to be shown next) across users but also facilitate smooth quality switches for a given user.

Fairness. Figures 18 and 19 plot the distributions of FPS and content quality respectively, for five smartphones. Note that although the instantaneous available bandwidth may differ across the devices, in the long run, each device largely gets an equal share of the bandwidth (as verified by us). Also, since each device replays multiple human users' motion traces, this should largely smooth out the impact of motion diversity among the human users. In addition, the devices' computational power heterogeneity is considered by the adaptive object quality selection mechanism (§3.6). Therefore, we expect the distributions to be similar among the devices. This is indeed shown in Figures 18 and 19, confirming that AQC can achieve a decent level of fairness among the devices.

Real Phones vs. Emulated Devices. We observe small performance differences between the two device groups: the 5 real smartphones and the 10 Raspberry Pis. Their average stall duration (across all users' sessions belonging to each group) differs by less than 2%; for both groups, 99% of the timer events have MFC=0; both groups also exhibit very similar FPS distributions; the median content quality is CRF 24.2 and 26.1 for the phone and the Pi group, respectively. This difference is likely attributed to the conservative emulation settings (*e.g.,* decoding latency) used in emulation. Overall, We believe that Firefly is accurately emulated on the Pis.

5.3 Micro Benchmarks

We now present several micro benchmarks to showcase the impact of key design decisions of Firefly.

Impact of AQC. Figure 20 plots the stall duration across all VR sessions with AQC enabled vs. disabled. When AQC is disabled, we consider two extreme cases: always fetching

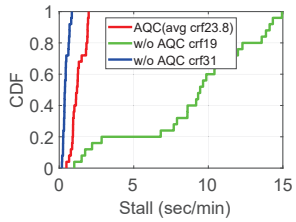

Figure 20: Impact of adaptive quality control (AQC).

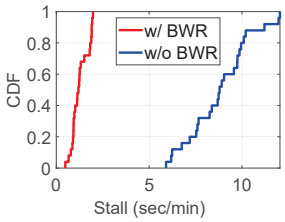

Figure 21: Impact of BW reservation in AQC.

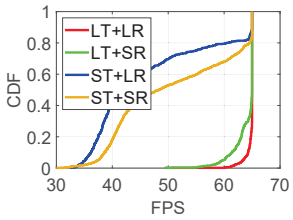

Figure 22: Impact of viewport prediction method.

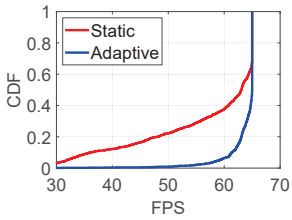

Figure 23: Impact of adaptive foreground object quality selection.

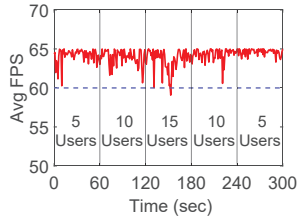

Figure 24: Impact of user dynamics on frame rate.

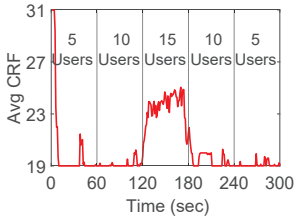

Figure 25: Impact of user dynamics on content quality.

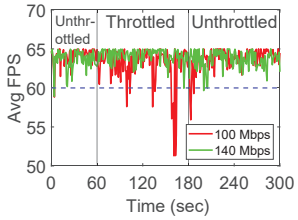

Figure 26: Impact of BW changes on frame rate.

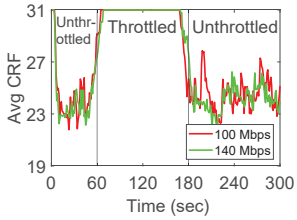

Figure 27: Impact of BW changes on content quality.

the highest quality (CRF=19) and always fetching the lowest quality (CRF=31). As shown, the former suffers from very long stalls (median: 9.6 sec/min); the issue with the latter is the low content quality (CRF 31). AQC instead strikes a much better tradeoff: the achieved average quality is CRF 23.8, while the stall duration is only slightly increased compared to statically using CRF=31 without AQC.

Impact of Bandwidth Reservation in AQC. Recall from §3.4 that we make an important design decision in AQC by reserving for each user a fixed amount of bandwidth to handle the user's sudden motion that may incur unexpected bandwidth utilization. Figure 21 indicates that this mechanism is highly beneficial. If bandwidth reservation (BWR) is disabled, the median stall duration increases drastically from 1.2 sec/min to 8.8 sec/min.

Impact of Viewport Prediction. To justify our viewport prediction design, we consider four variations shown in Figure 22. "LT+LR" is Firefly's approach where we use **L**inear regression (LR) for both the **T**ranslational movement and **R**otational movement prediction; "ST+SR" represents a naïve **S**tatic strategy: directly using the current viewport as the predicted viewport by assuming the user is stationary in both the translational and the rotational dimensions; "LT+SR" corresponds to using LR for translational prediction and Static for rotational prediction; "ST+LR" represents using Static and LR for translational and rotational prediction, respectively. Here, we consider all 25 users' motion traces by replaying them sequentially using one Samsung Galaxy Note 8 phone.

Figure 22 shows that Firefly's approach, LT+LR, achieves the overall highest FPS. Also, LT+SR significantly outperforms ST+LR and ST+SR. This suggests that translational prediction accuracy plays a more important role in determining the system performance compared to rotational prediction accuracy. The reason is that large tiles (90°×180°) can shield many rotational prediction errors (§3.3) but not any translational prediction error.

Impact of Adaptive Object Quality Selection. By analyzing the logs produced by the experiments in §5.2, we find that oftentimes many avatars indeed appear in the viewport: in more than 40% (10%) of the viewports, 4 (8) or more avatars need to be rendered, and this number can reach 10. Too many foreground objects incur high local rendering workload in particular for computationally weak devices. This overhead can be effectively mitigated by the object quality selection scheme (§3.6), which adaptively reduces the fidelity of foreground objects (in our experiments, the users' avatars) to maintain a high FPS. Figure 23 suggests that by disabling this feature (the "Static" curve, which always renders the objects at the highest quality), the FPS drops significantly: the fraction of 1-sec windows with <60 FPS increases from 8% to 37%.

5.4 Adaptiveness to Number of Users

We conduct an experiment to demonstrate that Firefly can properly handle users dynamically joining and leaving the system. We begin with 5 randomly chosen devices at t=0; at t=60s, 5 randomly chosen devices join the system; at t=120s, 5 more devices start their VR sessions; at t=180s, 5 devices leave the system; finally at t=240s, 5 more devices leave. Figures 24 and 25 plot the average FPS and average CRF across all users, respectively, over time. As shown, regardless of the user dynamics, the frame rate almost always stays above 60 FPS. Meanwhile, the content quality well adapts to the bandwidth available to each individual device. When there are no more than 10 devices, each device can enjoy the highest content quality at CRF 19. With 15 devices, AQC reduces the average quality level to ∼24 due to bandwidth scarcity while maintaining fairness across users (Figures 18 and 19). The fluctuations in Figures 24 and 25 (also in Figures 26 and 27 to be described in §5.5) are attributed to our averaging method (first over a 1-second window and then over all users) for

calculating each FPS and content quality sample.

5.5 Adaptiveness to Available Bandwidth

We conduct two experiments to investigate how Firefly adapts to changing network bandwidth. In the first one, we begin with unthrottled bandwidth (around 200 Mbps as reported by the AP) at t=0; we then use the Linux `tc` tool [12] to throttle the AP-wide bandwidth to 140 Mbps at t=60s; the bandwidth throttling is removed at t=180s. The second experiment is the same except that the bandwidth throttling is set to 100 Mbps. For both experiments, we fix the number of devices to 15.

Figures 26 and 27 plot the average FPS and average content quality across all users, respectively, over time. When the total bandwidth reduces, the content quality immediately drops to the lowest in order to maintain a high frame rate. For 140Mbps bandwidth throttling, AQC manages to stabilize the frame rate at 60+ FPS. For 100Mbps throttling, each device gets only ∼6.7Mbps bandwidth on average that can barely support even the lowest quality level at CRF=31. As a result, the frame rate occasionally drops below 60 FPS.

5.6 Comparison with MUVR

MUVR [47] is a recently proposed, state-of-the-art multi-user mobile VR framework. It is also (to our knowledge) the most relevant work to Firefly. In MUVR, a server maintains a centralized cache that stores the rendered and encoded VR content. Given a user's translational position, the server can directly transmit the view if it is cached; otherwise the server needs to render the view and properly cache it. In their evaluation, the authors emulated 4 concurrent users of MUVR.

We quantify the effectiveness of MUVR on our Office dataset using simulation. The setup is similar to §5.2 where we replay 15 randomly selected users' motion traces 5 times. Meanwhile, we simulate the centralized cache: for every frame, all devices simultaneously "send" their translational positions to the server; upon cache misses, the server will "render" the corresponding positions and add them to the cache. We assume the cache is initially empty and has unlimited capacity. We find that for 27% of the time, there are more than 5 concurrent cache misses, *i.e.,* the server needs to render for more than 5 devices. According to our pilot experiment in §2, this cannot be supported by even a high-end GPU, leading to poor scalability. Firefly eliminates this issue by performing *exhaustive* offline rendering (§3.1). It also introduces other important components that MUVR does not have such as AQC, viewport adaptation, and handling foreground objects.

5.7 Resource Usage and Thermal Overhead

CPU, GPU, and Memory. Firefly incurs acceptable runtime overhead and resource footprint on mobile devices. During a VR session, the CPU usage (reported by the Android Studio Profiler) is no higher than 30% across the five smartphones.[1] The overall memory usage (CPU+GPU) is no higher than 1.6 GB, which is mostly spent on L1 and L2 cache. Note that the cache capacities (L1, L2, and L3) are adjustable in Firefly.

Energy Usage and Thermal Characteristics. To profile the energy usage, we fully charge the five phones, and then repeat the experiment in §5.2 by running on each phone five back-to-back VR sessions. After that (25 minutes later), we record the remaining battery percentage, which ranges from 92% to 96% (average 93.8%) depending on the device's power consumption and battery capacity. We also monitor the CPU/GPU temperature. After continuously playing the VR content for 25 minutes, the highest temperature (either GPU or CPU) among the devices is 50°C, which only feels moderately warm. Overall we think the above energy and thermal characteristics are completely acceptable for mobile VR.

6 Related Work

360° Video Streaming. There exist a plethora of work on streaming 360° videos. Prior systems such as Flare [53], Rubiks [38], Freedom [60], and POI360 [62] also take a viewport-adaptive streaming approach. Some other studies focus on viewport prediction for 360° videos [24, 33, 39]. Compared to the work above, Firefly extends the viewport-adaptation idea to generic VR that involves both the rotational and translational viewport movement. In particular, we demonstrate how viewport adaptation can benefit multi-user VR systems.

Single-user Mobile VR has also been well investigated. Flashback [27] and Furion [44] demonstrate high-quality single-user VR on COTS smartphones. Flashback is a completely local system (on a single device, content stored in SD card). We leverage its core concept of offline rendering to support high-quality, networked multi-user VR. We extend Furion to a multi-user version and quantitatively compare it with Firefly in §5.2. MoVR [22, 23] employs 60 GHz mmWave wireless for mobile VR. Liu *et al.* [48] proposed system-level optimizations for the mobile VR rendering pipeline. Tan *et al.* explored supporting mobile VR over LTE [61]. None of the above work explicitly focuses on the multi-user scenario.

Multi-user VR/AR. Despite a plethora of work on single-user VR, much fewer studies have been conducted on its multi-user counterpart. The most relevant work to Firefly is MUVR [47] that is described in detail in §5.6. Bo *et al.* developed a multi-user 360° video streaming system based on multicast [25]. A recent positioning paper [49] discusses several practical issues of designing a multi-user VR system (without implementation). Some studies investigated multi-user or collaborative augmented reality (AR) [54, 55, 68]. Compared to the above work, Firefly is a generic multi-user VR system. It achieves much better scalability compared to MUVR.

[1] Android Studio Profiler does not report the GPU utilization.

7 Discussion

Efficient Offline Rendering. Future VR applications can involve large and complex scenes, drastically increasing the overheads of offline rendering. Firefly plans to explore well-known rendering optimization techniques [29, 50] which use different hierarchical structures for adapting to the surface tessellation and level of detail.

Handling Dynamic Background. While significantly boosting scalability, Firefly's offline rendering assumes the background content is static (but can be arbitrarily complex). Simple dynamic content involving short animation sequences can still be rendered offline. Complex dynamic content (involving lighting, reflection, *etc.*) has to be rendered at runtime.

Enhancing Firefly using Computer Graphics and Multimedia Techniques. While the contributions of Firefly are mostly on the system side, we are aware that Firefly can be enhanced by various techniques developed from the computer graphics and multimedia community. For example, the 3D models of foreground objects can be simplified using techniques proposed by [31, 32, 35, 56]; visibility or distance culling [34, 36, 52] can be applied to reduce the runtime rendering overhead while maintaining objects' visual qualities; more sophisticated partitioning [58, 59] can be employed to make caching more efficient for both static background and dynamic foreground; more efficient video codec such as H.265 [37] and the next-generation H.266 standard [1] can be leveraged to further reduce the bandwidth footprint for background content delivery; powered by recent advances in deep learning, deep neural networks (super-resolution) can be applied to enhance the image quality [30, 65]. The above approaches are orthogonal to Firefly's exhaustive rendering paradigm and are compatible with the AQC scheme.

Improving Motion Prediction. Firefly employs online linear regression for motion prediction. Despite being simple, it is experimentally demonstrated to be efficient and effective. The prediction accuracy could be further improved using more sophisticated prediction methods. For instance, over 3-DoF (degree-of-freedom) head movement data, deep learning approaches such as LSTM (Long Short-Term Memory) was found to outperform classic machine learning in particular when the prediction window is long [64]. Another promising direction is to enrich the feature set using, for example, velocity, acceleration, and even VR content features such as saliency [33]. We plan to explore the above directions in our future work.

7.1 Lessons Learned

We learned several important lessons from Firefly, which may guide the design of future multimedia systems.

First, Wirth's law [21] also applies to multimedia: the content resolution/quality increase may outpace the graphics technology evolution. While 3D computer games already use some pre-computation techniques such as projecting pre-rendered 2D panoramic background [2] and rendering faraway 3D objects as 2D sprites, we believe that more extensive offline computation and caching will remain a core technique that can scale up high-quality content rendering on commodity hardware, in particular in emerging multimedia services such as mixed reality and cloud gaming.

Second, scheduling content delivery in a multi-user system requires considering a wide range of factors: network bandwidth, device rendering capability, users' QoE, users' interaction, and cross-user fairness. Our experience of developing AQC indicates that while establishing a full-fledged optimization framework may be difficult, a robust heuristic-driven algorithm can work well in practice. In addition, to adapt to users' fast-paced, bursty interactions, the scheduling algorithm needs to run at a frequency that is much higher than traditional videos' bitrate adaptation algorithms [40, 41, 51, 66].

Third, from traditional videos (0 DoF) to 360° videos (3-DoF) and then to VR/volumetric (6-DoF), multimedia content tend to become more immersive and interactive. To embrace such trends, future multimedia systems need more intelligence, which is not limited to motion prediction as showcased in Firefly. Elements such as users' eye movement [43, 46], users' voice, salient visual content [33], and sound source, to name a few, can all be leveraged to infer viewers' intention and henceforth to facilitate system-level decision making such as content prefetching and scheduling.

Fourth, in addition to content, client devices, and server, the network (in particular, the wireless one) is also a key component whose interplay with the multimedia system needs to be carefully optimized. The lower-layer wireless channel information could be leveraged to guide network resource allocation. In Firefly, we demonstrate this over 802.11ac WiFi (§4). Similar cross-layer design could be conducted for other WiFi standards (802.11ax [26]) and cellular networks [62, 63].

8 Concluding Remarks

We have demonstrated with Firefly that it is feasible to support 15 VR users at 60 FPS using COTS smartphones and a single AP/server. Our design makes judicious decisions on (1) partitioning the workload (offline vs. runtime, client vs. server), (2) making the system adaptive to the available network/computation resources, both collectively and locally to each user, and (3) handling users' fast-paced, bursty motion. We believe that the core concepts of Firefly are applicable to other multi-user scenarios such as those of augmented reality and mixed reality.

ACKNOWLEDGEMENTS

We thank the voluntary users who participated in our study, the anonymous reviewers for their valuable comments, and Philip Levis for shepherding the paper. This work was supported in part by NSF Award #1903880 and #1915122.

References

[1] 3 New Codecs Coming in 2020. . https://nofilmschool.com/three-new-codecs-are-coming.

[2] An Adventure in Pre-Rendered Backgrounds. https://justinmeiners.github.io/pre-rendered-backgrounds/.

[3] Android MediaCodec API. https://developer.android.com/reference/android/media/MediaCodec.html.

[4] Blender. https://www.blender.org/.

[5] Bluetooth VR controller. https://www.amazon.com/VR-Bluetooth-Controller-Kasonic-Smartphones/dp/B01E7Z72NQ/.

[6] BYOD Popularity. https://www.forbes.com/sites/larryalton/2017/03/27/how-important-is-a-byod-policy-5-strategies-for-millennials/.

[7] Equirectangular Projection. http://mathworld.wolfram.com/EquirectangularProjection.html.

[8] Google AR and VR: Bringing pixels front and center in VR video. https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/.

[9] Google Cardboard. https://vr.google.com/cardboard/.

[10] How to Build a PC for Virtual Reality. https://www.logicalincrements.com/articles/vrguide.

[11] Keeping the virtual world stable in VR. https://www.qualcomm.com/news/onq/2016/06/29/keeping-virtual-world-stable-vr.

[12] Linux TC. http://man7.org/linux/man-pages/man8/tc.8.html.

[13] Museum Unity Asset. https://assetstore.unity.com/packages/3d/environments/museum-vr-complete-edition-89652.

[14] Next-generation video encoding techniques for 360 video and VR. https://code.fb.com/virtual-reality/next-generation-video-encoding-techniques-for-360-video-and-vr/.

[15] Oculus Rift. https://www.oculus.com/rift-s/.

[16] Office Unity Asset. https://assetstore.unity.com/packages/3d/environments/urban/qa-office-and-security-room-114109.

[17] Per-Title Encode Optimization. https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2.

[18] Samsung Gear VR. https://www.samsung.com/global/galaxy/gear-vr/.

[19] The very real health dangers of virtual reality. https://www.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html.

[20] What Is Per-Title Encoding? https://bitmovin.com/per-title-encoding/.

[21] Wirth's Law. https://www.techopedia.com/definition/24381/wirths-law.

[22] Omid Abari, Dinesh Bharadia, Austin Duffield, and Dina Katabi. Cutting the cord in virtual reality. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 162–168. ACM, 2016.

[23] Omid Abari, Dinesh Bharadia, Austin Duffield, and Dina Katabi. Enabling high-quality untethered virtual reality. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2017)*, pages 531–544, 2017.

[24] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1161–1170. IEEE, 2016.

[25] Yanan Bao, Tianxiao Zhang, Amit Pande, Huasen Wu, and Xin Liu. Motion-prediction-based multicast for 360-degree video transmissions. In *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2017.

[26] Boris Bellalta. Ieee 802.11 ax: High-efficiency wlans. *IEEE Wireless Communications*, 23(1):38–46, 2016.

[27] Kevin Boos, David Chu, and Eduardo Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 291–304. ACM, 2016.

[28] Shenchang Eric Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38, 1995.

[29] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Bdam – batched dynamic adaptive meshes for high performance terrain visualization. In *Computer Graphics*

Forum, volume 22, pages 505–514. Wiley Online Library, 2003.

[30] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020.

[31] Xavier Décoret, Frédo Durand, François X Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *ACM SIGGRAPH 2003 Papers*, pages 689–696. 2003.

[32] Carl Erikson and Dinesh Manocha. Gaps: General and automatic polygonal simplification. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 79–88, 1999.

[33] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation Prediction for 360 Video Streaming in Head-Mounted Virtual Reality. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72. ACM, 2017.

[34] Thomas A Funkhouser and Carlo H Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 247–254, 1993.

[35] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[36] Enrico Gobbetti and Fabio Marton. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM SIGGRAPH 2005 Papers*, pages 878–885. 2005.

[37] Dan Grois, Detlev Marpe, Amit Mulayoff, Benaya Itzhaky, and Ofer Hadar. Performance comparison of h. 265/mpeg-hevc, vp9, and h. 264/mpeg-avc encoders. In *2013 Picture Coding Symposium (PCS)*, pages 394–397. IEEE, 2013.

[38] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 482–494. ACM, 2018.

[39] Xueshi Hou, Sujit Dey, Jianzhong Zhang, and Madhukar Budagavi. Predictive View Generation to Enable Mobile 360-degree and VR Experiences. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 20–26. ACM, 2018.

[40] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of SIGCOMM 2014*, pages 187–198. ACM, 2014.

[41] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. In *Proceedings of CoNEXT 2012*, pages 97–108. ACM, 2012.

[42] Alan D Kalvin and Russell H Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3):64–77, 1996.

[43] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2017)*, pages 545–559, 2017.

[44] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Transactions on Mobile Computing*, 2019.

[45] Jiankun Li and C-CJ Kuo. Progressive coding of 3-d graphic models. *Proceedings of the IEEE*, 86(6):1052–1063, 1998.

[46] Tianxing Li and Xia Zhou. Battery-free eye tracker on glasses. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 67–82, 2018.

[47] Yong Li and Wei Gao. Muvr: Supporting multi-user mobile virtual reality with resource constrained edge cloud. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 1–16. IEEE, 2018.

[48] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–80. ACM, 2018.

[49] Xing Liu, Christina Vlachou, Feng Qian, and Kyu-Han Kim. Supporting untethered multi-user vr over enterprise wi-fi. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 25–30, 2019.

[50] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Siggraph 2004 Papers*, pages 769–776. 2004.

[51] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of SIGCOMM 2017*, pages 197–210. ACM, 2017.

[52] Soraia R. Musse, Christian Babski, Tolga Capin, and Daniel Thalmann. Crowd modelling in collaborative virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 115–123, 1998.

[53] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114. ACM, 2018.

[54] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. Avr: Augmented vehicular reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 81–95. ACM, 2018.

[55] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. Sharear: Communication-efficient multi-user mobile augmented reality. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 109–116, 2019.

[56] Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. In *Computer Graphics Forum*, volume 15, pages 67–76. Wiley Online Library, 1996.

[57] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. The computing landscape of the 21st century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2019.

[58] Gernot Schaufler and Wolfgang Stürzlinger. A three dimensional image cache for virtual reality. In *Computer Graphics Forum*, volume 15, pages 227–235. Wiley Online Library, 1996.

[59] Jonathan Shade, Dani Lischinski, David H Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 75–82, 1996.

[60] Shu Shi, Varun Gupta, and Rittwik Jana. Freedom: Fast recovery enhanced vr delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 130–141. ACM, 2019.

[61] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. Enabling Mobile VR in LTE Networks: How Close Are We? In *Proceedings of SIGMETRICS 2018*. ACM, 2018.

[62] Xiufeng Xie and Xinyu Zhang. Poi360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 336–349. ACM, 2017.

[63] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. pistream: Physical layer informed adaptive video streaming over lte. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 413–425, 2015.

[64] Tan Xu, Bo Han, and Feng Qian. Analyzing viewport prediction under different vr interactions. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 165–171, 2019.

[65] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018)*, pages 645–661, 2018.

[66] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of SIGCOMM 2015*, pages 325–338. ACM, 2015.

[67] Matt Yu, Haricharan Lakshman, and Bernd Girod. A framework to evaluate omnidirectional video coding schemes. In *Proceedings of the Symposium on Mixed and Augmented Reality (ISMAR) 2015*, pages 31–36. IEEE, 2015.

[68] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. Cars: collaborative augmented reality for socialization. In *Proceedings of the 19th International Workshop on Mobile computing Systems & Applications*, pages 25–30. ACM, 2018.

[69] Jingbo Zhao, Robert S Allison, Margarita Vinnikov, and Sion Jennings. Estimating the motion-to-photon latency in head mounted displays. In *2017 IEEE Virtual Reality (VR)*, pages 313–314. IEEE, 2017.