Mehrabi, Abbas; Siekkinen, Matti; Kämäräinen, Teemu; Ylä-Jääski, Antti

Multi-Tier CloudVR

Published in:
ACM Transactions on Multimedia Computing, Communications and Applications

DOI:
10.1145/3429441

Published: 01/06/2021

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
Other

Please cite the original version:
Mehrabi, A., Siekkinen, M., Kämäräinen, T., & Ylä-Jääski, A. (2021). Multi-Tier CloudVR: Leveraging Edge Computing in Remote Rendered Virtual Reality. *ACM Transactions on Multimedia Computing, Communications and Applications*, *17*(2), Article 49. https://doi.org/10.1145/3429441

Multi-Tier CloudVR: Leveraging Edge Computing in Remote Rendered Virtual Reality

ABBAS MEHRABI, Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, United Kingdom
MATTI SIEKKINEN, Department of Computer Science, Aalto University, Espoo, Finland
TEEMU KÄMÄRÄINEN, Department of Computer Science, University of Helsinki, Helsinki, Finland
ANTTI YLÄ-JÄÄSKI, Department of Computer Science, Aalto University, Espoo, Finland

The availability of high bandwidth with low-latency communication in 5G mobile networks enables remote rendered real-time virtual reality (VR) applications. Remote rendering of VR graphics in a cloud removes the need for local personal computer for graphics rendering and augments weak graphics processing unit capacity of stand-alone VR headsets. However, to prevent the added network latency of remote rendering from ruining user experience, rendering a locally navigable viewport that is larger than the field of view of the HMD is necessary. The size of the viewport required depends on latency: Longer latency requires rendering a larger viewport and streaming more content. In this article, we aim to utilize multi-access edge computing to assist the backend cloud in such remote rendered interactive VR. Given the dependency between latency and amount and quality of the content streamed, our objective is to jointly optimize the tradeoff between average video quality and delivery latency. Formulating the problem as mixed integer nonlinear programming, we leverage the interpolation between client's field of view frame size and overall latency to convert the problem to integer nonlinear programming model and then design efficient online algorithms to solve it. The results of our simulations supplemented by real-world user data reveal that enabling a desired balance between video quality and latency, our algorithm particularly achieves the improvements of on average about 22% and 12% in term of video delivery latency and 8% in term of video quality compared to respectively order-of-arrival, threshold-based, and random-location strategies.

CCS Concepts: • **Networks → Cloud computing**; **Mobile networks**; **Network resources allocation**; • **Computing methodologies → Virtual reality**; • **Mathematics of computing → Integer programming;**

Additional Key Words and Phrases: Multi-access edge computing (MEC), rendered virtual reality (VR), joint optimization, integer nonlinear programming (INLP), greedy algorithm

ACM Reference format:
Abbas Mehrabi, Matti Siekkinen, Teemu Kämäräinen, and Antti Ylä-Jääski. 2021. Multi-Tier CloudVR: Leveraging Edge Computing in Remote Rendered Virtual Reality. *ACM Trans. Multimedia Comput. Commun. Appl.* 17, 2, Article 49 (May 2021), 24 pages.
https://doi.org/10.1145/3429441

Authors' addresses: A. Mehrabi, Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, NE1 8ST, United Kingdom; email: abbas.mehrabidavoodabadi@northumbria.ac.uk; M. Siekkinen and A. Ylä-Jääski, Department of Computer Science, Aalto University, 02150 Espoo, Finland; emails: matti.siekkinen, antti.ylajaaski@aalto.fi; T. Kämäräinen, Department of Computer Science, University of Helsinki, Yliopistonkatu 4, 00100 Helsinki, Finland; email: teemu.kamarainen@helsinki.fi.

1551-6857/2021/05-ART49 $15.00
https://doi.org/10.1145/3429441

1 INTRODUCTION

The goal in **virtual reality (VR)** is the complete immersion of the user in a virtual world. This kind of experience is only possible if the visual experience is of high quality and navigation within the world is as seamless as in the real world. High-quality visual experience in a dynamic VR scene is nowadays possible only when a dedicated graphics card with hundreds or thousands of **graphics processing unit (GPU)** cores is used to render graphics. Traditionally, this is done using a **personal computer (PC)** that is cable connected to the VR headset. Cabling and required close proximity of the PC makes the solutions expensive and cumbersome to setup. Therefore, approaches to "cut the cord" have been investigated, ranging from using a wireless local link between the VR headset and PC to offloading all or most of the rendering to a remote GPU cloud alleviating the need for PC deployment altogether. In this article, we focus on the latter approach. In such systems, the client device receives a compressed video stream from the remote rendering server and transmits user controls, such as head motion, to the server.

Multi-access edge computing (MEC) aims to transfer the computing and communication resources from a far-away central sever to the network edges near the end consumers [13]. Mobile adaptive video streaming in conjunction with MEC technology has been investigated recently [20–22]. This prior work has focused on **quality of experience (QoE)** optimization of **video on demand (VOD)** services that stream static two-dimensional (2D) video content from either a further-away cloud or nearer edge server to mobile clients. Different from these studies, we consider in this article a real-time remote VR rendering service in a MEC environment. The idea is that MEC is able to bring rendering service with very short network latency but cannot necessarily always serve all clients because of limited amounts of GPU hardware. In contrast, a consolidated GPU cloud offers virtually unlimited amount of GPU resources but with a longer network latency. These two together can serve all clients but it is unclear what is the best strategy to divide the total workload between the two. In addition, MEC offers the ability to query the radio network status in real time, hence enabling bitrate adaptation at the edge. To this end, we design optimization techniques for maximizing the system delivered user experience given a set of VR clients.

Seamless navigation in the VR world requires that the view drawn on a VR display must react immediately when the user rotates her/his head, which in turn requires very short latency. However, it is possible to hide the latency inherent in remote rendering from the user [5, 15, 17, 29]. The specific solution we consider in this article is to render a locally navigable viewport that is larger than the **field-of-view (FOV)** of the user's headmounted display, as presented in Reference [15]. The larger viewport makes it possible to locally rotate the last received frame when the user's head rotates, which prevents the user from getting exposed to unrendered content before an updated frame is received from the server. This approach essentially reduces the perceivable latency associated with head rotations to a level obtained with local device operation only.

The size of the viewport required to conceal the effect of latency from the user depends naturally on the end-to-end latency as well as head motion dynamics. This means that when rendering in a further-away consolidated cloud, it is necessary to compute more compared to rendering at the network edge, given identical head movements. Furthermore, the amount of total content transmitted to client in turn affects latency. In other words, the larger the rendered scene with a constant resolution per visible area or the higher the quality of a video frame, the more they eat up network resources, which translates into longer latency when transmitted over a radio link with a specific over-the-air data rate. As wireless network and edge GPU are both bottleneck resources and the wireless link quality differs between clients, the problem translates into finding the best possible allocation of clients between edge and cloud as well as selecting streamed video bitrate for each client so that the quality of user experience is maximized.

To achieve the above mentioned goal, our optimizations aim to minimize the overall computational overhead and network traffic considering the amount of rendered content required to compensate for the resulting end-to-end latency. Although ideally the viewport size should be sufficiently large to conceal latency altogether from the user upon head rotations, it is still desirable to have as short end-to-end latency as possible from the user experience perspective as well to support smooth interactions using a handheld controller, for instance. That is why our optimization algorithms also strive for minimal overall latency. Optimizing VR streaming, or more appropriately 360° video streaming, has been studied earlier (e.g., References [1, 10, 26]). However, in our work we need to consider the computational constraints due to real-time rendering. Although different aspects of remotely rendered VR have been explored to some extent, combining both edge and consolidated cloud rendering is still largely to be investigated. Our contributions are therefore as follows:

- We present an edge accelerated system for real-time cloud rendering for VR. The cornerstone of the system is client to edge/cloud allocation with adaptive rendering and streaming that strives for maximal user experience.
- We formulate an optimization problem to maximize user experience in the proposed edge accelerated cloud rendering system. In particular, we propose joint optimization of clients perceived quality and latency subject to available processing resources at the edge.
- We design heuristic-based algorithms to solve the optimization problem and evaluate their performance through simulations supplemented by real-world user data.

2 RELATED WORK

In VOD streaming, several quality adaptation approaches for improving the QoE of mobile users have been proposed during the past few years [14, 24, 32, 33, 35, 38]. Some studies provide comprehensive survey on the factors that affect the QoE of mobile users in **dynamic adaptive video streaming over HTTP (DASH)** [3, 16, 25, 27]. The majority of VOD quality adaptation solutions are client based, in which the quality selection logic is purely run at the client side without collaboration with other network entities. Network-assisted solutions were designed in which the clients, servers, and the network collaborate with each other through message passing mechanisms to allocate fairly the video qualities among the competing clients as well as optimally utilize the limited resources of the network [8, 18].

The evolution toward the service-based 5G core network architecture, involving software-defined networking and network function virtualization, has also accelerated the design of network-assisted quality adaptation solutions. Bentaleb et al. [2] leverage software-defined networking to optimize video streaming QoE. Concerning edge computing, moving the network resources (processing, caching) to the edge using MEC unlocks the use of clients contextual information within radio access network, which in turn helps the client-side adaptation logic toward optimal/fair video quality and resource allocation [22, 34, 36]. Tran et al. [34] investigated the problem of joint collaborative caching and processing for adaptive bitrate video streaming at the network edge. Mehrabi et al. [22] proposed to use MEC for DASH services. In References [20, 21], the same authors extended the work with edge caching and device-to-device communication to optimize jointly users' QoE and network traffic.

Using MEC is more crucial in real-time video streaming due to the need for low latency, especially in interactive remote rendering applications, such as cloud gaming and remote rendered VR. Various strategies have been proposed in previous research to distribute the server infrastructure.

Choy et al. [6] conducted a large-scale measurement study to determine if existing cloud infrastructure can meet the requirements of the emerging class of latency-sensitive multimedia

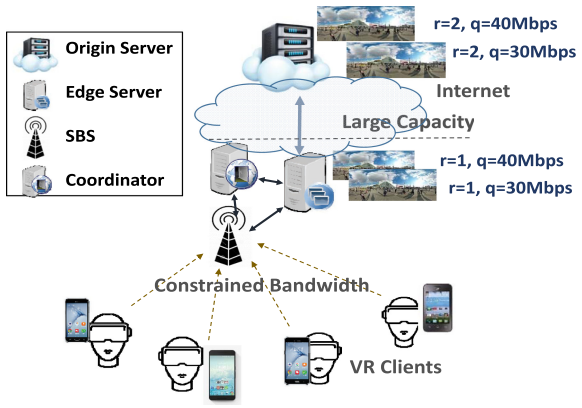


Fig. 1. MEC-assisted real-time VR rendering system.

applications. They showed that Amazon EC2 data center locations can provide a median latency of 80 ms to only 70% of the measured end users. Considering client and server-side delays, they state that a substantial increase is needed in the total number of data centers. Their solution is to leverage existing content distribution infrastructure by enhancing them with additional processing units and GPUs. This leads to an additional 28% of end-users who can meet the required latency requirements. In a follow-up work, Choy et al. [7] propose a hybrid infrastructure using both cloud and edge servers. They also show that careful game placement and server selection schemes can increase the number of end-users who can be served by the system.

The approach to render additional surroundings of the user to enable local navigation within a frame has been introduced in multiple papers [15, 28, 29]. Previous studies have also envisioned edge computing to enable a high-quality wireless VR experience [11, 12]. Shi et al. [29] also provide initial results that show the benefits of mobile edge rendered VR using a real-world prototype.

Compared to the related work, our work studies the use of edge computing specifically in remote rendering. The novelty lies in considering a multi-tier architecture including both edge and consolidated cloud and examining the benefits that edge offers over cloud only rendering. Our system is based on a specific method to anticipate head movements in VR usage to compensate for latency when the content is remotely rendered in real time. This combined with the multi-tier architecture gives rise to the problem formulation, which is original. Specifically, we derive an objective function to jointly optimize quality and latency for remote VR video rendering in the proposed cloud-edge server system, where the processing resources at the edge are limited, and solve it through heuristic-based algorithms.

3 EDGE COMPUTING ACCELERATED CLOUD RENDERING FOR VR

3.1 Remote Rendered VR

The computing and rendering power of smart phone-based and stand-alone VR devices is vastly inferior compared to desktop PC-based systems. Remote rendering can resolve this issue and enable high-quality VR content on stand-alone devices. In remote rendering applications, user input is sent from a thin client device to the server where the input is fed into the application logic with corresponding frames being sent back to the client as a compressed video stream. The thin client decodes the video stream and displays the frames on the end user device. Remote rendering has been mostly commercialized and researched in the form of cloud gaming, where the viewport of

a game, which is the visible area of the 3D world to the user, is captured and sent to the remote user.

Remote rendering applications add unavoidable latency to the end-to-end delay, which needs to be addressed in the system design. In cloud-rendered VR, the latency issues are even more highlighted. It has been estimated that the rotation latency from head movement to the corresponding response should be under 20 ms to avoid simulator sickness caused by the lag between the sensory inputs from visual and vestibular systems [23]. In addition to rotation latency, separate translation and interaction latencies are present in a VR application. Translation latency refers to the delay from a controller command to the user avatar moving in the virtual world and interaction latency as the delay when moving a game object by moving a hand-held controller. The different latencies are not equally important for user experience.

As even small amounts of rotation latency can incur motion sickness, the traditional approach of cloud gaming where only the visible viewport is rendered and sent to the thin client cannot be used without additionally rendering a portion of the surrounding world in the form of viewport scaling. In viewport scaling, additional area surrounding the user is rendered and streamed to the user, allowing the thin client to locally rotate the view based on the most recent head rotation. In the extreme case, a complete 360° view is sent for the user as a panoramic frame. This approach can mask the latency regarding head rotations in **three degrees of freedom (3DOF)** systems, where the head rotations control the rendering camera. A small-scale user study in Reference [15] suggests that it is effective in hiding the impact of latency from users even when rendering in a further-away located consolidated cloud server. Additional rendered surroundings do, however, increase the computational requirements of the server and can lead to unnecessary work and high bandwidth requirements. The amount of rendered extra viewport can however be optimized based on the current latency and head rotation speed.

3.2 System Overview

Figure 1 illustrates a schematics view of edge computing assisted real-time remote VR system. We consider a single-server MEC system that is associated with **small cellular base station (SBS)** from where the downlink radio resource blocks are **proportionally fair (PF)** allocated to the connected clients at each time slot. The VR clients join and leave the video streaming session at some random time slots. According to the orientation of the headset, the client sees some portion of the whole 360° scene, which we refer to as FOV. To compensate for rotation latency, the server renders a viewport that is larger than the FOV of the client so that the user has margin for head rotations before the next frame arrives. To optimize computational as well as network resources, the desired viewport size is just enough to compensate for the latency upon motion but not larger. To know what that size should be, we derive a relationship for the viewport size and latency based in user data in Section 3.6 that will also be used in the evaluation. At each time slot, the scene corresponding to the viewport can be rendered either at the cloud or at the edge.

In contrast to the wireless links with constrained capacity at the edge, we assume that the wired path connecting the network edge to the consolidated cloud is mainly fiber with large capacity and, hence, is not the bottleneck. We also assume that the delay to deliver video content from the cloud server to network edge is on average the same (may vary around the mean). However, the video delivery latency from edge to the client depends on client link quality, resources allocated to the client by SBS, and the size of rendered video data. The overall latency is defined as the total time of delivering a frame of rendered video from cloud or edge server to the client. We exclude the processing delays associated with client device or server as they do not depend on the server location (edge vs. cloud). The available processing resources of cloud servers are sufficient to render graphics for all clients at all times. In contrast, the processing resources of edge servers

are limited. The limited edge processing resources imply that rendering for some clients has to be performed at origin server due to the saturation of resources at the edge. Although we consider a single edge server for the sake of model simplicity, the same operations can be replicated to multi-server systems without any change to the underlying model and analytical derivations.

We also assume that the time taken to render a frame of video is same at either the cloud server or at an edge server, and, therefore, we do not explicitly account for it in the optimization problem. In other words, the frame rate is constant, and we require edge servers to be capable of rendering frames at the required rate for each served client. However, the edge rendering capacity is defined in terms of the output, i.e., total volume of encoded video per time unit. In this way, we make an implicit assumption that the rendering quality is linked to video encoding quality (bitrate) so that if a lower-quality video is delivered to a client, the graphics rendering process is automatically adjusted in a corresponding manner to avoid consuming unnecessary GPU resources.[1]

The centralized coordinator obtains the clients contextual information (arrival/departure time slots, wireless link quality) and the edge server processing status. It then solves a joint optimization problem to decide on the optimal video quality allocation to connected VR clients at each time slot. Therefore, bitrate adaptation is purely handled by the coordinators in contrast to traditional HTTP-based streaming systems in which it is often the client that runs the adaptation logic. Concerning the bandwidth fluctuation, it should be noted that the drop in network bandwidth simply causes the reduction in real-time video bitrate assigned to the client and as a result the degradation of average video quality of client during its VR session.

3.3 System Notations

We consider the scheduling of S mobile VR clients in a single server MEC system during $|T|$ discrete time slots where each slot has the fixed duration of $\Delta t = 1s$. The available bandwidth in time slot t at SBS associated with edge server is represented by $W^{(t)}$. The arrival and departure time slots of client $s \in S$, i.e., the time that client starts and finishes its VR session, are represented by A_s and D_s, respectively. The **signal to noise ratio (SNR)** received by VR client s at time slot t is represented by $SNR_s^{(t)}$. The theoretical throughput of client s at time slot t is denoted by $T_s^{(t)}$, which is computed according to the following approximation of Shannon upper bound [19]:

$$T_s^{(t)} = \begin{cases} 0; & if SNR_s^{(t)} < SNR_{min} \\ \alpha \cdot log_2\left(1 + 10^{\frac{SNR_s^{(t)}}{10}}\right); & if SNR_{min} \leq SNR_s^{(t)} < SNR_{max}, \\ T_{max}; & if SNR_s^{(t)} \geq SNR_{max} \end{cases} \tag{1}$$

where path loss parameter $\alpha = 0.6$ and SNR_{min}, SNR_{max}, and T_{max} are set to respectively -10 dB, 23 dB, and 4.4 bps/Hz according to the LTE downlink specifications reported in Reference [19].

The effective throughput is then obtained as follows: $Thr_s^{(t)} = (\frac{T_s^{(t)}}{\sum_{j \in S, j \neq s} T_j^{(t)}}) \cdot W^{(t)}$.

The delivered video to the client during its session is divided into f_{ps} fixed number of frames such that the frame size vary depending on the overall video size. The perceived latency of video delivery from the edge to client s with channel bandwidth $C_s^{(t)}$ at time slot t is represented by $l_s^{(t)}$. We also denote by $b_s^{(t)}$ the size of rendered video data of client s at time slot t.

The video frames are encoded and can be delivered to the client in M different qualities represented by set $R = \{q_1, q_2, \ldots, q_M\}$. The rendered viewport size and the allocated video quality of

[1]Some game engines allow directly adjusting the graphics quality of rendering process, whereas another possibility is to control a quality related factor, such as rendering resolution. However, in this work, we do not take a stance on how exactly it should be done to provide as high visual experience as possible with optimized GPU load and leave it for future work.

client s at time slot t are represented by respectively $r_s^{(t)}$ and $q_s^{(t)}$. The rendered viewport size is a unitless variable, and it is a factor of the FOV in user's HMD, while video quality has the unit of Mbps. Further, the variable video delivery delay from the origin server to edge server as network jitter at time slot t and the maximum tolerable latency by a VR client are represented respectively by $d^{(t)}$ and t_{max}. The edge server has the available processing resources of $p^{(t)}$ (in Gb) at time slot t and constant ϕ is also defined, which states the processing weight of edge server. The binary decision variables $x_{se}^{(t)}$ and $x_{sc}^{(t)}$ are also defined, which indicate the rendering of VR scene for client s at the edge and cloud server, respectively, at time slot t.

3.4 Average Video Quality

The average bitrate at which the rendered scene is encoded into video indicates the quality perceived by the client. Although the relation between the video bitrate and the perceived quality may not be linear in practice, we consider the direct mapping between them for the sake of model simplicity. The average video quality of client s during its VR session is given by $Q_s = (\sum_{t=A_s}^{D_s} q_s^{(t)})/(D_s - A_s)$.

Since server switching can negatively impact the viewing experience of VR client, we also define a penalty term each time server switching happens. In other words, suppose $\Psi_s^{(t)}$ denotes the average switching frequency caused by VR client s up to time slot t during its video streaming session. The following relation is obtained: $\Psi_s^{(t)} = (\sum_{t'=A_s+1}^{t} \mathbb{I}(x_s^{(t')} \oplus x_s^{(t'-1)} = 1))/(t - A_s)$, where $\mathbb{I}(.)$ is the unity function. With the above relation, the average quality penalty during the video streaming session of VR client s is obviously equal to $\Psi_s = \sum_{t=A_s+1}^{D_s} \Psi_s^{(t)}$.

3.5 Average Latency

The overall latency for a client at each time slot depends on the location (the edge or origin server) from where the scene is rendered at that time slot. If the video is rendered at a cloud server, then it adds a delay of $d^{(t)}$ to the latency of video frame delivery to the client at time slot t. The latency associated to the video frame delivery from the edge to client s at time slot t is given by $l_s^{(t)} = b_s^{(t)}/(f_{ps} \cdot C_s^{(t)})$.

It is noteworthy to mention that with a fixed amount of video data, increasing the number of frames in the video implies the lesser amount of data per frame to be transmitted over the wireless link at each time slot. This in turn implies a reduction in video delivery latency per time slot that is consistent with the relation $l_s^{(t)}$ given above. Considering both edge and cloud rendering, the average latency during VR session of client s is obtained as $L_s = (\sum_{t=A_s}^{D_s} l_s^{(t)} + x_{sc}^{(t)} \cdot d^{(t)})/(D_s - A_s)$.

3.6 Size of Rendered Viewport

We also need to know the relationship between latency and the required viewport size to compensate such latency upon motion. In other words, we want to find out how large the frustum needs to be so that the user does not notice its limited size, yet keeping it constrained for resource consumption (computing and bandwidth) reasons. We first derive the mathematical notations that are independent on how this relationship is characterized. After that, we exemplify one way of deriving this relationship from real user data that we have used in the system evaluation. We discuss the limitations and possible extensions to this way of characterizing the required viewport size in Section 7. Similarly, while in this work we only consider rotational motion, i.e., 3DOF, we discuss in that section how 6DOF could be supported.

From the mathematical point of view, we express this relationship by defining a single variable function f_r, which states the viewport size in terms of latency as follows: $r_s^{(t)} = f_r(l_s^{(t)} + x_{sc}^{(t)} \cdot d^{(t)})$.

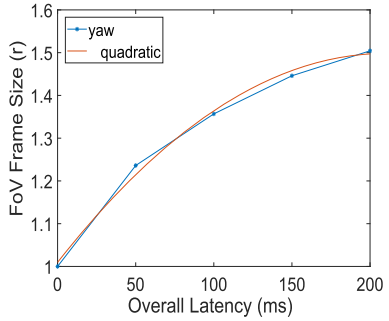


Fig. 2. Quadratic approximation for the relationship between FOV and latency.

The volume of a video frame (in bits) to be delivered to a client depends on the viewport size and the video encoding parameters, specifically quantization, which determines the bitrate and quality tradeoff. The volume of rendered video data by client s at time slot t is given as follows: $b_s^{(t)} = f_b(r_s^{(t)}, q_s^{(t)})$, where f_b is a two-variable function that maps the viewport size and the video quality to the volume of video data that should be transmitted to the client over wireless link. Combining the above obtained relations $r_s^{(t)}$ and $b_s^{(t)}$, we can express the size of rendered video data in terms of latency and quality as follows: $b_s^{(t)} = f_b(f_r(l_s^{(t)} + x_{sc}^{(t)} \cdot d^{(t)}), q_s^{(t)})$.

Now, from the equation for $l_s^{(t)}$ obtained in Section 3.5, we derive the following relation between latency and video quality: $l_s^{(t)} = f_b(f_r(l_s^{(t)} + x_{sc}^{(t)} \cdot d^{(t)}), q_s^{(t)})/(f_{ps} \cdot C_s^{(t)})$.

To characterize function f_r, which states the viewport size in terms of latency, we analyzed head movements from two VR usage datasets presented in References [9, 37], similarly to our earlier work in Reference [15]. We calculated samples indicating the total amount of rotation observed around the y-axis (pitch) and z-axis (yaw) within a specific time window, i.e., latency. From this, we inferred the fraction of peripheral faces (in addition to the front face that is always rendered entirely) that need to be rendered so that the user's head movements according to the datasets would not exceed the viewport size before a new frame arrives. We plot the 99th percentile of these samples as a function of latency in Figure 2. The curve tells us the viewport size that is sufficient in 99% of time to compensate for rotations given specific amount of latency according to the datasets. In here, we only plot yaw, because it is more prevalent in the traces. Pitch can be analyzed in the same way, but for simplicity, in this article, we assume that rotations around the horizontal axis exhibit the same relationship. The figure also shows a polynomial curve that we fit to these samples. In this way, we characterize the relationship as a quadratic function $r = f_r(l) = p_1 l^2 + p_2 l + p_3$. Furthermore, we derive function $b = f_b(r, q) = 4(r - 0.5)^2 \cdot q$ for mapping the viewport size and video quality (bitrate) to the video stream volume (in Mbps).

4 JOINT OPTIMIZATION PROBLEM FOR MAXIMAL USER EXPERIENCE

The problem of jointly optimizing average video quality and latency for each individual client $s \in S$ is formulated as the following **mixed integer nonlinear programming (MINLP)** optimization model:

$$\underset{x_{se}, x_{sc}, q_s}{Maximize} \quad \alpha w_1 Q_s - (1 - \alpha) w_2 L_s - w_3 \Psi_s, \tag{2}$$

subject to

$$C_s^{(t)} + \sum_{\forall j \in S, j \neq s} C_j^{(t)} \leq W^{(t)}, \qquad \forall 1 \leq t \leq |T|, \tag{3}$$

$$x_{jc}^{(t)} \cdot d^{(t)} + l_s^{(t)} \leq t_{max}, \qquad \forall 1 \leq t \leq |T|, \tag{4}$$

$$x_{se}^{(t)} \cdot \phi f_b \left(f_r \left(l_s^{(t)} + x_{jc}^{(t)} \cdot d^{(t)} \right), q_j^{(t)} \right) + \sum_{\forall j \in S, j \neq s} x_{je}^{(t)} \cdot \phi f_b \left(f_r \left(l_s^{(t)} + x_{jc}^{(t)} \cdot d^{(t)} \right), q_j^{(t)} \right) \leq p^{(t)}, \tag{5}$$

$$\forall 1 \leq t \leq |T|$$

Equation $l_s^{(t)}$ *obtained in Section 3.6*

$$x_{se}^{(t)} + x_{sc}^{(t)} = 1, \qquad A_s \leq t \leq D_s, \tag{6}$$

$$q_s^{(t)} \in R, \quad x_{se}^{(t)}, x_{sc}^{(t)} \in \{0, 1\}, \qquad \forall 1 \leq t \leq |T|. \tag{7}$$

In the above MINLP problem, the variables $q_s^{(t)}$, $x_{se}^{(t)}$, and $x_{sc}^{(t)}$ are the only decision variables, $l_s^{(t)}$ is the dependent variable, and the values of other parameters are known in advance. The weighting parameter $0 \leq \alpha \leq 1$ is defined in the objective function to control the quality–latency tradeoff. Since the video quality and latency have different scales and units, we further define the parameters w_1, w_2 in the objective function to balance the quantitative values of video quality and latency. Depending on the scale of video quality, the actual values of these two parameters are set later in the simulation results. It is noted that w_1 is unitless while w_2 takes the unit of Mb/s^2. Further, we define the weighting w_3 as a coefficient of third term in the objective function that accounts for server switching penalty and scales up in accordance with the average server switching frequency value Ψ_s. Parameter w_3 has the unit of Mbps and we investigate its impact later in simulations.

Constraint (3) ensures that the summation of allocated bandwidth to client s and the other connected VR clients at each time slot does not exceed the available bandwidth. Constraint (4) guarantees that the total latency perceived by client s at each time slot is no more than the upper bound. It is noted that this constraint is enforced to make sure that one time slot scheduling of clients is feasible in our system. Constraint (5) ensures that the summation of workload caused by client s and the workload of other connected VR clients at the edge does not exceed the rendering capacity of edge server. The equation $l_s^{(t)}$ that states the relation between latency and video quality is also added to the set of constraints. Constraint (6) states that each client must be served either by the edge or by the cloud. Finally, constraints (7) determine the range of decision variables.

It is worth pointing out that the size of optimization problem (2)–(7) depends on the number of VR clients and the number of deployed edge servers in the system. More precisely, the optimization problem has the size of $|T| \cdot (2S + S + S)$ where S is the number of VR clients in the system. Here, the first term is for the number of quality and latency decision variables (variables $q_s^{(t)}$ and $l_s^{(t)}$) for every client in all time slots, the second term is the number of decision variables for clients to cloud allocation during all time slots (binary variables $x_{sc}^{(t)}$. Note that there is only one cloud server) and the third term is the number of decision variables for clients to edge server allocation during all time slots (binary variables $x_{se}^{(t)}$. Note that there is only one edge server in the system).

5 ONLINE HEURISTIC SOLUTIONS

The problem formulation (2)–(7) belongs to the class of NP-hard problems due to the existence of integer decision variables in the model. The problem formulation is also non-convex due to the combination of both integer and continues constraints in the model. Using offline solutions

such as the combination of integer relaxation and branch and bound technique can find the optimal solutions to the offline problem (2)–(7). But these techniques suffer from high complexity, and also the implementation of offline solutions is not applicable in practice, since the information of VR clients are not available in advance. However, some techniques may try to relax the set of discrete constraints to continues ones and then solve the resulted problem. However, designing effective heuristics to round the continues variables to the original integer ones is a challenging task, since the returned solutions should guarantee some degree of closeness to the optimal solutions.

To overcome the above challenges, we design in this section online greedy-based algorithms to obtain high-quality solutions with low computational complexity. In other words, our solutions are based on greedy algorithms that directly solve the decision variables and are implemented in an online manner, which make them feasible for practical applications.

Before designing the algorithms, we convert first the MINLP problem (2)–(7) to an INLP problem by taking advantage of the equation $l_s^{(t)}$, which was obtained in Section 3.6. More precisely, we derive the latency of the client at each time slot in term of its video quality at that time slot by solving the equation $l_s^{(t)}$ in term of q. Suppose g is the function mapping the video quality to the latency, i.e., $l_s^{(t)} = g(q_s^{(t)})$, which is derived by solving the equation $l_s^{(t)}$ from Section 3.6. Replacing latency using the quality function g, the MINLP problem (2)–(7) for each individual client $s \in S$ is converted to the following INLP problem:

$$\underset{x_{se}, x_{sc}, q_s}{Maximize} \quad \alpha w_1 Q_s - (1-\alpha) w_2 g(Q_s) - w_3 \Psi_s, \tag{8}$$

subject to

$$C_s^{(t)} + \sum_{\forall j \in S, j \neq s} C_j^{(t)} \leq W^{(t)}, \qquad \forall 1 \leq t \leq |T|, \tag{9}$$

$$g(q_s^{(t)}) + x_{sc}^{(t)} \cdot d^{(t)} \leq t_{max}, \qquad \forall 1 \leq t \leq |T|, \tag{10}$$

$$x_{se}^{(t)} \cdot \phi f_b(f_r(g(q_s^{(t)}) + x_{sc}^{(t)} \cdot d^{(t)}), q_s^{(t)}) + \sum_{\forall j \in S, j \neq s} x_{je}^{(t)} \cdot \phi f_b(f_r(g(q_j^{(t)}) + x_{jc}^{(t)} \cdot d^{(t)}), q_j^{(t)}) \leq p^{(t)}, \tag{11}$$

$$\forall 1 \leq t \leq |T|$$

$$x_{se}^{(t)} + x_{sc}^{(t)} = 1, \qquad A_s \leq t \leq D_s, \tag{12}$$

$$q_s^{(t)} \in R, \quad x_{se}^{(t)}, x_{sc}^{(t)} \in \{0, 1\}, \qquad \forall 1 \leq t \leq |T|. \tag{13}$$

Next, we introduce two variations of low complexity online greedy-based algorithms to solve the optimization problem, both of which are designed to be run by a centralized coordinator. Note that the coordinator is edge location (i.e., base station) specific, which means that the approach scales to larger systems by replicating the coordinators as more edge locations are added.

5.1 Link-based Greedy Quality Assignment

The first proposed algorithm is called **Link-based Greedy Quality Assignment (LGQA)**. The name comes from the fact that it specifically considers the wireless link quality of each clients in decision making. The rationale is that it is preferable to serve a client with a relatively poor link quality from the edge. The reason is that a lower latency of edge rendering allows smaller viewport size, which is especially important when the wireless link quality is poor and each bit transmitted consumes more network resources compared to a situation when the wireless connectivity is good.

ALGORITHM 1: Link-based Greedy Quality Assignment (Run by the centralized coordinator)

1: **Input:** $|T|$, S, R : Number of time slots, number of VR clients, set of available video qualities
2: **Output:** Binary allocations $x_{se}^{(t)}$, $x_{sc}^{(t)}$ and the integer video quality $q_s^{(t)}$ for each client $s \in S$ and time slot $1 \leq t \leq |T|$
3: **for** each time slot $1 \leq t \leq |T|$ **do**
4: **Allocate** RBs to connected clients according to PF policy;
5: **Compute** the effective throughputs $Thr_s^{(t)}$ (using the equations given in Section 3.3) for all clients $s \in S$ such that $A_s \leq t \leq D_s$;
6: **Sort** the clients in increasing order of throughput for all clients put;
7: **Insert** the sorted clients into set S';
8: **for** each client $s \in S'$ **do**
9: $maxObjective = 0$;
10: **for** each video quality $q \in R$ **do**
11: **if** $q \leq Thr_s^{(t)}$ **then**
12: $x_{se}^{(t)} = 1$; $x_{sc}^{(t)} = 0$;
13: **if** the allocation of q satisfies constraints (10), (11)
14: **AND** $\alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi > maxObjective$ **then**
15: $maxObjective = \alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi$;
16: $q_s^{(t)} = q$; $temp_{se}^{(t)} = 1$; $temp_{sc}^{(t)} = 0$;
17: $x_{se}^{(t)} = 0$; $x_{sc}^{(t)} = 1$;
18: **if** the allocation of q satisfies constraint (10)
19: **AND** $\alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi > maxObjective$ **then**
20: $maxObjective = \alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi$;
21: $q_s^{(t)} = q$; $temp_{se}^{(t)} = 0$; $temp_{sc}^{(t)} = 1$;
22: $x_{sc}^{(t)} = temp_{sc}^{(t)}$; $x_{se}^{(t)} = temp_{se}^{(t)}$;
23: **if** $x_{se}^{(t)} == 1$ **then Update** $p^{(t)}$;
24: **if** for each $q \in R$: $q > Thr_s^{(t)}$ **then**
25: $x_{se}^{(t)} = 1$; $x_{sc}^{(t)} = 0$;
26: $q_s^{(t)} = q_{min}$; **Update** $p^{(t)}$;
27: **Return** $x_{se}^{(t)}, x_{sc}^{(t)}, q_s^{(t)}$ $\forall s \in S, 1 \leq t \leq |T|$;

The pseudo-code of the algorithm is presented in Algorithm 1. At each time slot and in online manner, the radio resources of SBS are first allocated to the connected VR clients in proportionally fair manner (line 4). After computing the effective throughput (line 5), the algorithm then sorts the clients in increasing order of their throughput (wireless link quality) at the current time slot (lines 6 and 7). The rationale behind such pre-ordering of clients is that those with poor link quality are prone to obtain higher video qualities with less perceived latency when allocated to the edge. The algorithm then checks the available video qualities (in set R) in a greedy manner and decides on the location (edge or cloud server) and the most suitable video quality that maximizes the client's objective function (8) and satisfies the constraints, i.e., constraints (10) and (11), if the client is assigned to edge (lines (12)–(15)) and only constraint (10) if the client is assigned to the cloud (lines (16)–(18)). Mathematically speaking, the allocated quality to client $s \in S$ at time slot t, i.e., $q_s^{(t)}$ after sorting the clients is obtained by solving the following local optimizer:

$$q_s^{(t)} = \underset{\forall q \in R}{arg\,\textbf{maximize}} \{\alpha w_1 Q_s^{(t)} - (1-\alpha) w_2 g(Q_s^{(t)}) - w_3 \Psi_s^{(t)}\}, \tag{14}$$

subject to constraints (9)–(13)

where $Q_s^{(t)}$ and $\Psi_s^{(t)}$ are respectively the average video quality obtained and the average server switching penalty incurred by client s up to (including) time slot t.

At each time slot, the rendering resources of edge server are updated (line 19). Finally, if the effective throughput of the client at that time slot is below every video quality in set R, then the client is then assigned to edge server at that time slot with minimum video quality (lines 20 and 21).

5.1.1 Computational Complexity. It is seen from Algorithm 1 that at every time slot, the allocation of video quality to the active clients is performed by first sorting the clients based on their link quality and then checking all the possible video qualities and corresponding latency, which is obtained by solving equation $l_s^{(t)}$ from Section 3.6.

In the worst case, S clients are active at every time slot and the sorting of clients based on link quality can be performed in $O(S \cdot log(S))$. Then for each client, the allocation of $|R|$ available video qualities at two different locations (edge and cloud) should be checked in the greedy manner. Also, solving the equation $l_s^{(t)}$ using bisection method for finding the corresponding latency for each video quality takes $T_{bisection}$ time. In the case that the effective throughput of client falls below every available video quality assuming with probability p, the allocation of client to the edge with minimum video quality is performed in $O(1)$. Now, considering the total number of $|T|$ streaming time slots, the above complexities together yield: $T_{LGQA} \in O(|T| \cdot S \cdot (log(S) + 2|R|T_{bisection} + p))$

It is seen that $T_{bisection} \in O(log(t_{max}/\delta))$ where t_{max} is the upper bound on the latency perceived by VR clients and δ is the error tolerance in solving equation $l_s^{(t)}$ using bisection method. Since $p \leq 1$ and $T_{bisection} >> p$, it results in the following worst-case time complexity for LGQA algorithm: $T_{LGQA} \in O(|T| \cdot S \cdot (log(S) + 2|R|log(t_{max}/\delta))) = O(|T| \cdot S \cdot log(S(t_{max}/\delta)^{2|R|}))$.

5.2 Augmented Greedy Quality Assignment

The second variant of a greedy algorithm uses a different logic for the allocation of video qualities to clients at each time slot. We term it **Augmented Greedy Quality Assignment (AGQA)**, and the rationale is to select the most appropriate video quality and the optimal location (edge or cloud) for the client at each time slot, which yields higher utility (objective value (2)) with lower edge rendering occupation. The pseudo-code of the algorithm is presented in Algorithm 2.

More formally, the AGQA algorithm aims to improve the solutions of LGQA on objective value (8) by first constituting the set of all feasible quality levels and corresponding latency from both edge and cloud at each time slot. The algorithm then sorts all quality and latency pairs for each client in decreasing order of $\frac{\alpha w_1 q - (1-\alpha) w_2 l - w_3 \psi}{p}$. Here, variables q, l, ψ, and p are respectively the quality, the corresponding latency, the server switching penalty, and the required rendering resources when a particular quality q is allocated to the client. After sorting, the algorithm allocates the first quality level in the sorted list to each client. Then, the rendering location is decided. The edge rendering capacity is occupied each time the selected video quality is rendered at the edge server until edge rendering capacity is fully saturated. Finally, those clients that have not been assigned any quality level are allocated to the edge server with minimum quality level at the current time slot.

5.2.1 Computational Complexity. We now derive the worst-case computation complexity of AGQA algorithm. The Algorithm 2 suggests that finding the latency corresponding to maximum $2|R|$ feasible quality levels (from both edge and cloud server) using bisection method takes $O((2S \cdot |R|)log(t_{max}/\delta))$ time at every time slot and for at most S active VR clients. Then, the worst-case time complexity of sorting set S' in decreasing order of ratio between the utility value and resource consumption is of order $O((2S \cdot |R|)log(2S \cdot |R|))$. Finally, the allocation of quality levels to clients according to the sorted list S' (including the allocation of those clients with no feasible quality level to edge with probability $p < 1$) takes $O(2S \cdot |R|)$ time. Now, putting all the above complexities together and with $|T|$ time slots, we obtain the following: $T_{AGQA} \in O((2|T| \cdot S \cdot |R|)log(2S \cdot |R| \cdot t_{max}/\delta))$.

In practical scenarios, the time complexity of both LGQA and AGQA algorithms grow with the order of $S.log(S)$ when the number of VR clients, S, increases, which is obviously less than the

ALGORITHM 2: Augmented Greedy Quality Assignment (Run by the centralized coordinator)

Input: $|T|$, S, R : Number of time slots, number of VR clients, set of available video qualities
Output: Binary allocations $x_{se}^{(t)}$, $x_{sc}^{(t)}$ and the integer video quality $q_s^{(t)}$ for each client $s \in S$ and time slot $1 \leq t \leq |T|$
Create empty set S';
for each time slot $1 \leq t \leq |T|$ **do**
  Allocate RBs to connected clients according to PF policy;
  Compute the effective throughputs $Thr_s^{(t)}$ (using the equations given in Section 3.3)
    for all clients $s \in S$ such that $A_s \leq t \leq D_s$;
  for each client $s \in S$, $A_s \leq t \leq D_s$ **do**
    for each video quality $q \in R$ **do**
      if $q \leq Thr_s^{(t)}$ **then**
        Compute corresponding latency $l = g(q)$
          when $x_{se}^{(t)} = 1$;
        if allocation of q satisfies constraint (10) **then**
          Append value $\frac{\alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi}{\phi q}$ to set S';
        Compute corresponding latency $l = g(q)$
          when $x_{sc}^{(t)} = 1$;
        if allocation of q satisfies constraint (10) **then**
          Append value $\frac{\alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi}{\phi q}$ to set S';
  Sort set S' in decreasing order of $\frac{\alpha w_1 q - (1-\alpha) w_2 g(q) - w_3 \psi}{\phi q}$;
  for each client $s \in S$, $A_s \leq t \leq D_s$ **do**
    Select the first quality q in set S';
    if the corresponding $x_{se}^{(t)} == 1$ **AND**
      constraint (11) is satisfied **then**
        Allocate $q_s^{(t)} = q$; **Update** $p^{(t)}$;
    if the corresponding $x_{sc}^{(t)} == 1$ **then**
        Allocate $q_s^{(t)} = q$;
  for each client $s \in S$, $A_s \leq t \leq D_s$ **do**
    if $q_s^{(t)} == \emptyset$ **then**
      $x_{se}^{(t)} = 1$; $q_s^{(t)} = q_{min}$; **Update** $p^{(t)}$;
Return $x_{se}^{(t)}, x_{sc}^{(t)}, q_s^{(t)}$ $\forall s \in S, 1 \leq t \leq |T|$;

quadratic order. Therefore, the algorithms scale with the number of clients reasonably well when considering larger scale deployments.

We also note that the proposed greedy-based algorithms fail to return feasible solutions under the scenario when the consumed wireless resources by the allocation of minimum video quality to all VR clients exceed the available resources at SBS. This could happen due to the poor wireless link quality of clients; for instance, they are physically located far from the SBS.

5.3 Optimality Analysis

In this section, the optimality performance of the proposed quality allocation algorithms are discussed. More precisely, we aim to derive lower bound on the solutions returned by algorithms with respect to the optimal solutions of problem (8)–(13). Since analyzing the optimality gap of the algorithms for this general problem is not straightforward, we first study the optimality of AGQA algorithm for a special case of the problem. We have detailed the optimality analysis of AGQA algorithm for the special case of problem (8)–(13) in the appendix. Later, we study the optimality performance of LGQA algorithm through simulations.

Indeed, the approximation factors are some rigorous lower bounds on the solutions generated by our algorithms. However, deriving a strict upper bound on the solutions returned by our algorithms, which is less useful compared to the lower bound, is not straightforward. It is also worth pointing out that the proposed algorithms do not work based on any iterative procedure which incur high complexity. In regard to convergence to the sub-optimal solutions, our algorithms

provide guaranteed approximation bound with respect to the optimal solutions (optimality gap) as evidenced by rigorous analytical analysis given in appendix as well as in Section 6.12.

6 EVALUATION

In this section, we perform simulations supplemented with real-world user data to evaluate the performance of two proposed algorithms compared to some baseline solutions.

6.1 Experiment Setup

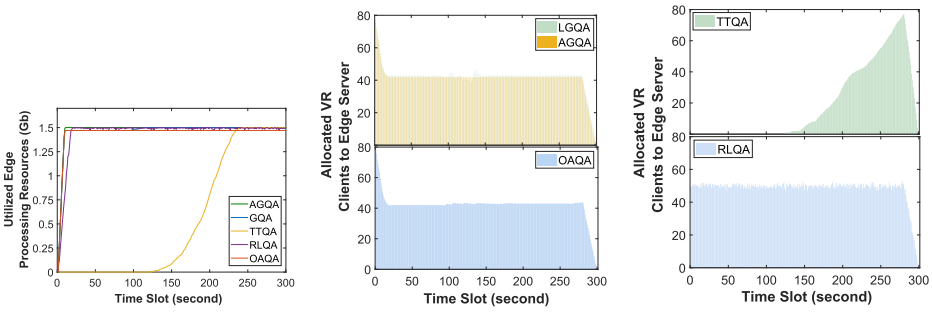
We consider a single server MEC system and video streaming of $|S| = 100$ mobile VR clients during $|T| = 300$ time slots where $\Delta t = 1s$. Although we study a single edge location system for simplicity reasons, we note that the results are applicable also to a larger scale system having several edge locations. As link quality affects the optimization, we obtain downlink SNR traces of mobile clients using the full-fledged simulator SimuLTE [31]. The available resource blocks of SBS at each time slot is also fixed at 100. The effective throughput of the clients are then obtained using the equations given in Section 3.3. Unless explicitly mentioned, the arrival time of VR clients is randomly chosen from the uniform interval $U[0, 20s]$. Also, the clients remain active in the system until the end of their session, which is equal to the simulation time, i.e., they have the same constant departure time.

To make the optimization feasible, we define 10 discrete bitrates (and qualities) for the real-time video streamed to the clients: $R = \{10, 12, 15, 20, 22, 25, 28, 30, 32, 35$ Mbps$\}$. The viewport size is bound between $1 \leq r \leq 1.5$ and the upper bound of the latency is set to $t_{max} = 0.2s$. The number of delivered video frames per time slot is fixed at $f_{ps} = 60$, which provides a reasonably good VR experience. We scale viewport size with latency according to the relationship derived in Section 3.6.

As default, the delay of video frame delivery from origin server to the edge is constant at 50 ms across the scheduling time slots. In addition, we consider the default server switching penalty of $w_3 = 0$ in the following simulations, while the impact of both origin to edge server latency variation and varying the server switching penalty are investigated in Sections 6.9 and 6.10.

The bisection method [4] with error tolerance of $\delta = 0.001$ is also implemented to solve the equation $l = g(q)$. Unless explicitly mentioned, the edge can render graphics worth of 1.5 Gb per second of compressed video (i.e., per time slot) while, the cloud processing resources are sufficient to support the rendering graphics for all clients. We also set the optimization weighting parameter $\alpha = 0.5$ unless mentioned otherwise. The quantitative scaling parameters in the objective function are also set to $w_1 = 1$ and $w_2 = 1$ Mb/s^2. Finally, we note that the results presented are averages taken over 20 runs of simulation with confidence interval of 95%. We compare the performance of AGQA and LGQA against the following video quality assignment strategies.

- **Order of Arrival Quality Assignment (OAQA):** Using this strategy, the clients who arrive to the system first are served from the network edge. Once the edge processing resources are saturated, the subsequent clients are served from the cloud. At both locations, the graphics are rendered and encoded with the highest possible sustainable quality.
- **Throughput Threshold Quality Assignment (TTQA):** In this strategy, the VR client is assigned to the edge if its effective throughput is below some pre-defined threshold, otherwise, it is assigned to the origin server. The allocated video quality to the client at either the edge or origin server is decided using the greedy approach. We consider the default throughput threshold of 25 Mbps unless mentioned otherwise.
- **Random Location Quality Assignment (RLQA):** In this strategy, the VR client is randomly assigned to either the origin or edge server. After the server assignment, the output video quality is selected using the greedy heuristic.



(a) Edge Processing Utilization (b) A/LGQA and OAQA (c) TTQA and RLQA

Fig. 3. (a) Comparison between five strategies in term of average edge server processing utilization per time slot and allocated clients to edge server using (b) AGQA, LGQA, and OAQA and (c) TTQA and RLQA strategies.

6.2 Edge Processing Utilization

We first study the edge processing resources used over time (Figure 3(a)). The results indicate that after the first few seconds, all the strategies lead to a similar constantly full utilization of the edge resources except for TTQA. TTQA causes the edge resources to be used increasingly toward the end of the simulated time period. The reason is that it uses the effective throughput as the threshold to decide on the allocation between edge and cloud, and this threshold starts to make a difference toward the end of the simulated period where the link qualities become worse. So, in a sense it is an artifact of the simulated scenario.

Figure 3(b) and (c) show the patterns of client allocation to edge during the simulations. The patterns are qualitatively similar with all strategies except for TTQA. The reason is the same reason as in the case of edge utilization described above.

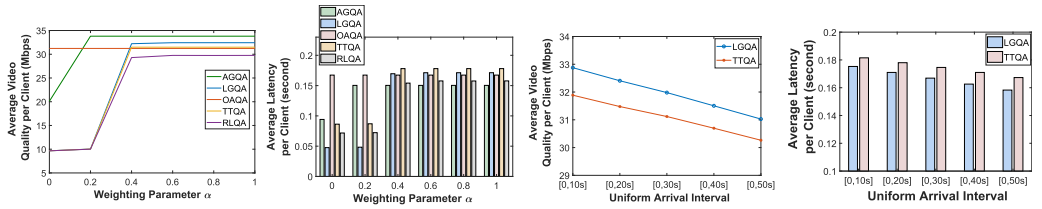
6.3 Video Quality

Next, we investigate the average video quality delivered to clients using different strategies. Figure 4(a) shows the results as a function of weighting parameter α that controls the optimization target.

We note that the OAQA strategy allocates a consistent video quality to the clients during whole video streaming session. In contrast to OAQA, which cannot adopt to clients preference, the other strategies offer different video qualities depending on whether quality or latency is preferred. We also note that RLQA solution results in lowest average video qualities among all the strategies. LGQA slightly improves the average quality (roughly 1 Mbps higher bitrate) compared to OAQA when $\alpha \geq 0.4$ by taking into account not only the alpha value but also the link quality of clients. AGQA further improves the average bitrate by about 6.65 Mbps compared to LGQA by sorting the clients in term of utility/processing ratio before the bitrate selection.

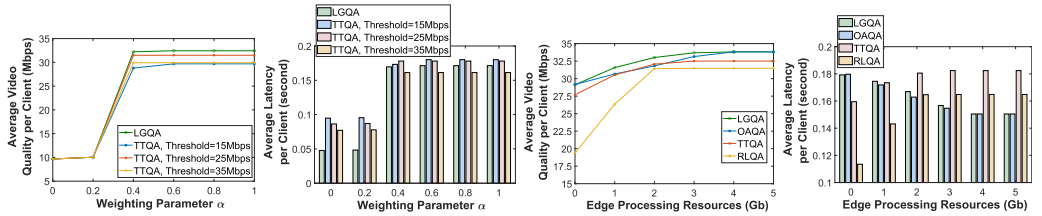
6.4 Latency

Figure 4(b) plots results pertaining to latency with the different strategies. As expected, RLQA, TTQA, and LGQA result in latency that is in accordance to the weighting parameter α. In other words, a lower latency is obtained with smaller values of parameter α and vice versa. AGQA's utility/processing ratio sorting of clients helps achieve the lowest average latency when $\alpha \geq 0.4$. The throughput-threshold server location logic used by TTQA also cannot efficiently optimize for latency as it causes higher latency compared to our algorithms. The reason is that selecting the



(a) Video Quality (b) Delivery Latency (c) Video Quality (d) Delivery Latency

Fig. 4. Comparison between five strategies in term of average (a) video quality and (b) latency per client and the impact of arrival interval on (c) average video quality and (b) average video delivery latency per client.



(a) Video Quality (b) Delivery Latency (c) Video Quality (d) Delivery Latency

Fig. 5. Impact of throughput threshold in TTQA algorithm on average (a) video quality and (b) latency per client and the impact of edge processing resources on average (c) video quality and (d) latency per client.

rendering location (edge or cloud) merely based on the throughput threshold results in situations when some clients with poor link quality are allocated to the cloud server. This in turn increases the latency that the clients perceive when they download high video qualities from the cloud server. Furthermore, although LGQA causes a bit higher latency compared to OAQA and RLQ when $\alpha \geq 0.4$, it results in higher average video quality for those alpha values.

Overall, the latency differences are notable: LGQA improves latency on average by 22% and 12% compared to OAQA and TTQA strategies, respectively. Although RLQA provides latency that is on par with our optimization algorithms, it cannot deliver similar video quality with that latency. Finally, we observe that AGQA reduces latency on average by 12% compared to LGQA when $\alpha \geq 0.4$. As a conclusion, AGQA algorithm outperforms in overall all other quality assignment solutions in terms of both average video quality and latency per client.

6.5 Impact of Arrival Interval

As the performance of some of the strategies is dependent on how the clients arrive to the system, we now turn our attention to that. To do this, we consider five different uniform intervals for the arrival time slot and a fixed alpha value $\alpha = 0.5$.

The results with LGQA and TTQA algorithms are shown in Figure 4(c) and (d). Increasing the interval length decreases the average video quality and latency with both algorithms. The reason is that as the arrival interval duration increases, more clients are simultaneously being served, which reduces resources available per client. With lower effective throughput, our algorithm allocates lower video quality to clients and with fixed weighting parameter α, it also results in lower average latency. However, with lower effective throughput on average and fixed throughput threshold, TTQA algorithm allocates the majority of clients to the edge, which results in lower average latency.

6.6 Impact of Throughput Threshold in TTQA Algorithm

We next investigate the impact of different throughput thresholds on the performance of TTQA strategy and compare the results to those achieved with our optimization algorithms. The results for three thresholds, 15, 25, 35 Mbps, are plotted in Figure 5(a) and Figure 5(b).

The results indicate that increasing the value of throughput threshold in TTQA algorithm may either increase or decrease the average video quality as the relationship between the throughput threshold and average perceived video quality per client is not straightforward. But, the average latency slightly decreases by increasing the throughput threshold. The reason is that as the throughput threshold increases, TTQA allocates larger number of clients to edge server and the average video quality that clients obtain merely depends on clients link quality and the available processing resources at the edge. However, using a higher throughput threshold in TTQA helps to perform most of video rendering tasks at edge server and, hence, reduce the average latency. We also note that overall LGQA performs better than TTQA with all the three threshold values. It is noted that we compared TTQA against LGQA algorithm rather than AGQA due to the fact that superiority of AGQA over LGQA has been justified as discussed in Sections 6.3 and 6.4.

6.7 Impact of Edge Processing Resources

We next investigate the impact of increasing the edge processing resources on the performance of algorithms. We simulated scenarios with different amounts of edge processing resources and $\alpha = 0.5$. The results are shown in Figure 5(c) and Figure 5(d).

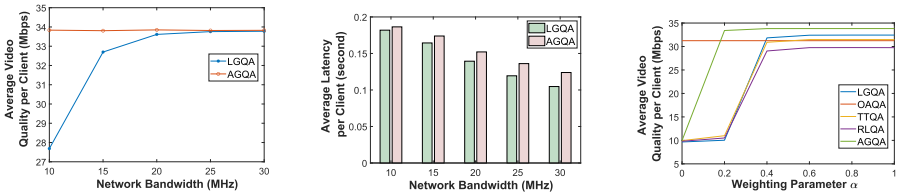
Increasing edge processing resources helps to improve the average video quality per client up to a certain point, regardless of the strategy used. This result is expected, since more processing power at the edge allows more clients to be served from the edge, and those clients can get a higher-quality video stream, because the shorter latency compared to cloud enables compensating for the latency using a smaller viewport size. The reason why 5 Gb of edge resources no longer improves the situation compared to 4 Gb of resources is simply that 4 Gb of resources is sufficient to serve all the clients from the edge considering the constraint on their wireless link quality.

Also average latency can be reduced by increasing the amount of edge processing power when using LGQA and OAQA algorithms. Similar to video quality, this stems from the fact that with more edge processing power, more clients are served from the edge, which provides a shorter latency than the cloud. In contrast, TTQA and RLQA behave in a different way as the average latency tends to increase when more processing power is allocated to the edge. The reason is that although TTQA and RLQA utilize the greedy video quality allocation logic, the allocation of clients to edge/origin server is performed in somewhat an uncontrolled way using these approaches.

The zero Gb worth of resources corresponds to the cloud only deployment case. Hence, in the simulated scenarios, edge acceleration can improve both video quality and latency by at most 16% compared to cloud only setup. The threshold value of 4 Gb worth of resources beyond which we no longer gain in terms of video quality and latency depends obviously on system parameters and cannot be a basis for decision making on dimensioning edge capacity for this type of services. However, perhaps a more interesting takeaway is that the curves are not linear but instead sublinear, which suggests that majority of gains are achieved well before the threshold limit is reached.

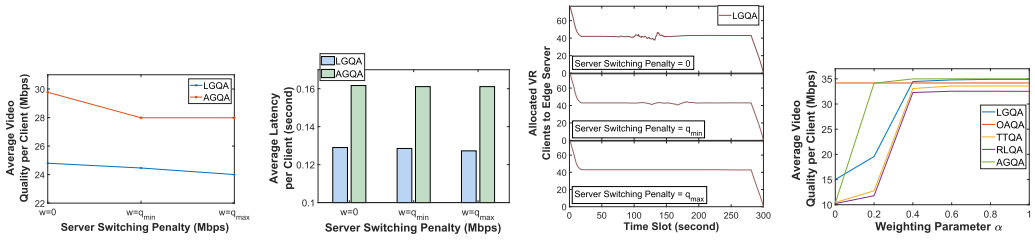
6.8 Impact of Available Network Bandwidth

Next, we investigate the impact of varying the available network bandwidth. To do this, we have varied the network bandwidth from 10 to 30 MHz and have shown the corresponding average video quality and delivery latency per client in Figure 6(a) and Figure 6(b), respectively.



(a) Average Quality (b) Average Latency (c) Average Quality

Fig. 6. Impact of network bandwidth on (a) average video quality and (b) average video delivery latency per client and (c) comparison between algorithms under uniform latency from origin to edge server.



(a) Average Quality (b) Average Latency (c) Edge Server Allocation (d) Average Quality

Fig. 7. Average (a) video quality and (b) latency per client and (c) the pattern of allocated clients to edge server under different server switching penalty. (d) Average video quality under millisecond bandwidth fluctuation.

As we can see from the results, the average video quality returned by LGQA algorithm increases with higher available bandwidth, which is due to the fact that the performance of this algorithm depends on wireless link quality of VR clients. However, the available bandwidth does not have much impact on average video quality of clients when using the AGQA algorithm. The reason is that this algorithm assigns high quality for video rendering of VR clients regardless of the location, i.e., the rendering could be at origin or edge server despite of some increase in latency.

Considering the average video delivery latency, we observe from the results in Figure 6(b) that with higher available bandwidth and better channel quality, the clients perceive lower latency. It is also observed that despite of no change in average video quality of AGQA algorithm, the clients perceive lower latency with high bandwidth under this algorithm. However, the latency of AGQA algorithm is higher than LGQA, which is due to the allocation of high video qualities to the clients.

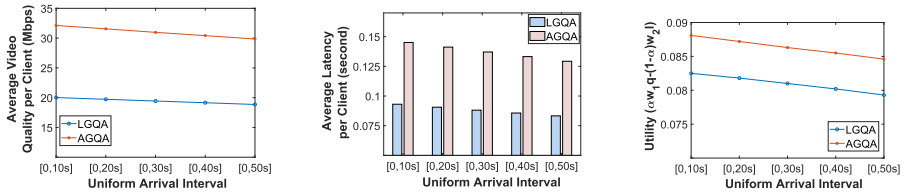
6.9 Origin to Edge Server Latency Variation

To evaluate the impact of origin to edge server latency variation, we have compared the proposed algorithms with baseline solutions when the origin to edge server latency is a random variable chosen from the uniform interval $U[40\text{ ms},60\text{ ms}]$. The comparison results in terms of average video quality per client for different values of parameter α have been illustrated in Figure 6(c).

As it is observed from the results, the proposed algorithms outperform the baseline solutions in terms of average video quality per client for different values of weighting parameter α. In turn, this reveals the robustness of the proposed algorithms under stochastic origin to edge server latency.

6.10 Server Switching Effect

We have further investigated the impact of server switching on average video quality and delivery latency per VR client. For the purpose of this simulation, we have considered three different values



(a) Average Quality (b) Average Latency (c) Average Utility

Fig. 8. Comparison between LGQA and AGQA in terms of average (a) video quality, (b) latency, and (c) utility per client for special case of video quality assignment problem.

for penalty weighting w_3 in the objective function (2) with setting $\alpha = 0.5$. For each weighting value, the corresponding average video quality and latency per client using our algorithms have been shown in Figure 7(a) and (b), respectively. Also, the pattern of number of clients allocated to edge server under different switching penalty has been illustrated in Figure 7(c).

As we can see from the results, by increasing the switching penalty, average video quality reduces, since it prevents the clients to switch more frequently for obtaining higher video quality. However, the reduction in average latency per client is negligible compared to reduction in video quality meaning that server switching has mainly impact on average video quality per client under the given setting. Although the reduction in average latency is negligible for this case, we achieve more noticeable reduction in average latency when optimization parameter α approaches to zero, because in such situations, the optimization considers higher priority on the latency term in the objective function than the quality term. As a result, it yields a more noticeable reduction in average latency by increasing the quality switching penalty.

As the observation from the results in Figure 7(c), less server switching is visible where the pattern of allocated clients to edge gets smoother with increase in switching penalty specifically between time slots 100 and 200. Similarly, it should be noted that we observe larger fluctuation in the number of clients assigned to edge server under smaller server switching penalty when the weighting parameter α approaches 1. The reason is that when α approaches 1, the optimization considers higher priority on the quality term in the objective function, and, therefore, the clients will have more switching between the servers under lower server switching penalty to achieve high video qualities.

6.11 Fine-grained Bandwidth Fluctuation

In the above simulations, we have considered the time slot scale of bandwidth fluctuation every 1 s. To show the robustness of the proposed model under fine-grained bandwidth fluctuation scale, we have further compared the proposed quality allocation algorithms with the baseline solutions when network bandwidth fluctuates every 1 ms with uniform distribution. For different values of weighting parameter α, the comparison results have been shown in Figure 7(d) in terms of average video quality per VR client.

As we can see from the results, the same pattern of improvements is observed as before, which in turn confirms the robustness of our proposed algorithms under fine-grained bandwidth fluctuation.

6.12 Optimality Performance through Comparison with AGQA

In this section, we aim to derive an approximate upper bound on the solutions of LGQA for the special case of video quality assignment problem (optimization problem (8)–(13)), which was discussed in Section 5.3. To achieve this, we compare LGQA with AGQA in terms of average video

quality, average latency, and utility per client. Figure 8 plots these metrics computed over all values of the alpha parameter (α from 0 to 1) and edge processing resources (from 0 to 5 Gb).

The figure tells us that, for this special case of the problem, AGQA delivers about 60% higher video bitrate on average compared to LGQA. At the same time, it causes about 35% longer average latency and outperforms LGQA by only 6% in terms of average utility per client. According to the analytical derivations in Appendix, AGQA achieves the approximation factor of $min(1/2, 1 - \epsilon)$, which implies that LGQA yields the approximate upper bound of $min(1/2, 1 - \epsilon)/1.06$ (where $0 < \epsilon < 1$ such that $(3 - 2\epsilon)\epsilon \leq 1/S$) for the considered optimization problem. With regard to our discussions, it is obvious that under the case when $\epsilon < 1/2$, algorithm AGQA achieves the approximation factor of 1/2, which means the solutions returned by AGQA algorithm for the special case of problem are greater than or equal to 1/2 of optimal solutions. This implies that the solutions returned by LGQA algorithm are greater than or equal to $0.5/1.06 \approx 0.47$ of optimal ones.

7 DISCUSSION AND FUTURE WORK

Concerning real-world deployment, the edge accelerated system for real-time cloud rendering for VR in this article requires real-time streaming and as such is not compatible with existing on-demand video services. However, the system could be integrated directly into cloud gaming services that have emerged in recent years (Google Stadia, Nvidia GeForce Now, Microsoft xCloud) provided that the required edge computing infrastructure is in place. Edge computing deployments are expected to happen in larger scale in conjunction with the deployment of stand-alone 5G networks.

The optimization scheme presented in this article allocates resources fairly to all users and applications. The system could be further extended to take into account the latency requirements of different application and game types and even the temporal variations in latency requirements between different phases of a VR application. Moreover, the viewport size–latency relationship that we characterized with the help of external datasets could also be application specific and something that the system could learn online or at least refine based on observed behaviour. The system model itself can accommodate any type of characterization of this relationship.

In this work, we focus on 3DOF use cases, which is reflected by how the the viewport size – latency relationship is characterized. Remote rendering with 6DOF VR headsets requires also the ability to reproject, or 3D warp, the incoming video frames to the new viewpoint upon translational motion [30]. The proposed system could be extended to support 6DOF by adding two features: (1) by streaming depth in addition to the RGB video (i.e., RGB-D) and (2) by considering motion in lateral, vertical, and longitudinal directions in addition to rotations when characterizing the viewport size–latency relationship. The latter feature stems from the observation that to produce a 3D warped view that fills the entire FOV, a larger than FOV size viewport is necessary.

We consider latency and visual quality as the two main factors that affect the quality of experience for a user in a cloud-rendered application compared to a locally run application. However, it should noted that there are some other factors that may affect the QoE of clients. Some of these could be controlled by the system, such as framerate and resolution, while others not, such as application engagement, VR headset specifics (FOV size, inherent latency, etc.), and user environment. Considering the other factors that could be controlled by the system is a potential avenue for extending the proposed system.

Concerning the optimality gap of the proposed algorithms, it should be noted that in large-scale VR remote rendered systems, the algorithms preserve the similar approximation factor based on the analysis given in the appendix despite of increase in the number of clients. It is also noteworthy to mention that although the optimality performance of algorithms is analyzed for a special case,

we expect that algorithms achieve slightly bigger approximation factor for general video quality assignment problem. Finally, we consider in this work that the centralized coordinator allocates the video quality to the VR clients merely with the objective of maximizing average video quality and minimizing the latency. This may compromise fairness in video quality allocation among the clients that compete for the shared bandwidth. Designing network-assisted solutions that also provide some level of fairness in quality allocation among the competing clients can be considered as another interesting future work.

8 CONCLUSION

This article focuses on the use of multi-access edge computing to accelerate remote cloud rendering of interactive VR. The core idea is that edge provides a limited amount of rendering resources with lower latency than cloud that has unlimited resources with longer latency. To cope with latency in remotely real-time-rendered VR, a viewport that is larger than the client device's FOV is rendered. The required size of the viewport depends on the latency and it affects the amount of video data that needs to be streamed to the client. We solve the problem of allocating clients to edge vs. cloud when edge resources are constrained and to optimize the resulting video quality, at which the graphics are rendered and streamed to the client, as well as latency in a parameter controlled way.

Due to the complexity, we utilize the relationship between the size of client's viewport and the perceived latency to simplify the optimization. Then, we design two variations of greedy-based algorithms for video quality assignment problem and further analyze their optimality performance. Through simulations using SNR traces and a quadratic approximation of viewpoint size and latency relationship obtained through real data from VR headset usage, we show that the proposed algorithms outperform baseline solutions in terms of average video quality and latency per client. Our results further reveal how increasing the amount of edge processing resources affects the average video quality and latency.

APPENDIX

A OPTIMALITY ANALYSIS OF AGQA ALGORITHM FOR SPECIAL CASE OF PROBLEM (8)–(13)

To proceed with the special case of problem, suppose P is the available processing resources of edge server at a given time slot and the client $s \in S$ with allocated quality level q_s at that time slot consumes the amount of $p_s = \phi q_s$ processing power. We define the special case of the problem considering the following assumptions.

- All the clients are served from the edge server that has the limited processing resources.
- For every client $s \in S$, the required edge processing power satisfies the condition $p_s \leq \epsilon P$ where $0 < \epsilon < 1$ is a small constant such that $(3 - 2\epsilon)\epsilon \leq 1/S$.

Note that deriving a constant approximation factor for AGQA algorithm that holds for all problem (special case) instances is infeasible. Therefore, we relate the optimality bound of algorithm to the structure of the problem by defining the small constant ϵ in the second assumption.

At each time slot, the set of available quality levels for each client depending on its link quality corresponds to multiple variations of each item in 0/1 KP. The achievable utility of joint quality and latency for each quality level of client equals the profit obtained by inserting that variation of item into the knapsack in 0/1 KP. Furthermore, the amount of consumed edge resources by allocating a quality level to the client is equivalent to the weight of corresponding variation of item in KP. Allocating the minimum quality level to a client (e.g., its link capacity is below the minimum available quality in set R) is also matched with no selection of any variation of corresponding item

in KP. Now, the problem of selecting quality levels for all the clients at the given time slot in our problem with the objective of maximizing joint quality and latency subject to the available edge processing resources translates to the problem of selecting maximum one variation of each item with the objective of maximizing the obtainable profit subject to the knapsack capacity in 0/1 KP.

We show now that AGQA algorithm achieves an approximation factor of $min(1/2, 1-\epsilon)$ for the special case of video quality assignment problem where $0 < \epsilon < 1$ is a small constant such that $(3-2\epsilon)\epsilon \leq 1/S$. Suppose $\{q_1, q_2, \ldots, q_j\}$ is the set of quality levels allocated to first j clients until the edge processing resources are fully utilized. Obviously, the remaining clients are allocated with the minimum quality level due to the lack of edge processing resources. Furthermore, suppose set $\{(u_1, p_1), (u_2, p_2), \ldots, (u_j, p_j)\}$ indicates the associated utility (objective value) and edge processing resources corresponding to the allocated quality levels. To proceed with the proof, we need to show that the following inequality holds: $u_1 + u_2 + \ldots + u_j \geq min(1/2, 1-\epsilon)OPT$.

Where OPT is the utility value obtained by the optimal algorithm for the special case of video quality assignment problem. Since AGQA algorithm allocates the quality levels to the clients in decreasing order of utility/processing ratio, that means the inequality $u_r/p_r \geq u_t/p_t$ for all clients $1 \leq r \leq j$ and $j+1 \leq t \leq S$ holds. This in turn implies the following:

$$\frac{u_1 + u_2 + \cdots + u_S}{p_1 + p_2 + \cdots + p_S} \geq \frac{u_{j+1} + u_{j+2} + \cdots + u_S}{p_{j+1} + p_{j+2} + \cdots + p_S}, \tag{15}$$

which is further simplified to the following inequality:

$$u_1 + u_2 + \cdots + u_S \geq (p_1 + p_2 + \cdots + p_S) \cdot \left(\frac{u_{j+1} + u_{j+2} + \cdots + u_S}{p_{j+1} + p_{j+2} + \cdots + p_S}\right). \tag{16}$$

It is straightforward that $p_1 + p_2 + \cdots + p_S \geq P$, because otherwise, the solution returned by AGQA is optimal, which completes the proof. From this, the inequality (16) yields the following:

$$u_{j+1} + u_{j+2} + \cdots + u_S \leq (u_1 + u_2 + \cdots + u_S) \cdot (p_{j+1} + p_{j+2} + \cdots + p_S)/P. \tag{17}$$

Now, since under the special case of problem, $p_s \leq \epsilon P$ for every client $1 \leq s \leq S$, we obtain the following inequality from Equation (17):

$$u_{j+1} + u_{j+2} + \cdots + u_S \leq (S-j)\epsilon(u_1 + u_2 + \cdots + u_S). \tag{18}$$

According to the pigeon hole principle, one of the following two conditions must hold:

$$u_1 + u_2 + \cdots + u_j \geq \frac{OPT}{2} \qquad u_{j+1} + u_{j+2} + \cdots + u_S \geq \frac{OPT}{2}. \tag{19}$$

Otherwise, $u_1 + u_2 + \cdots + u_S \leq OPT$, which is a contradiction, since the optimal utility cannot be more than the utility value obtained by allocating the quality levels to all the clients. If the first (left side) inequality in Equation (19) holds, then the algorithm achieves an approximation factor of 1/2. However, if the second inequality (right side) in Equation (19) holds, then, by combining inequalities (18) and this second inequality, we obtain the following: $u_1 + u_2 + \cdots + u_j \geq \frac{OPT}{2} \cdot \frac{1-(S-j)\epsilon}{(S-j)}$.

Since $(3-2\epsilon)\epsilon \leq 1/S$, we have $\frac{OPT}{2} \cdot \frac{1-(S-j)\epsilon}{(S-j)\epsilon} \geq (1-\epsilon)OPT$.

From these last two inequalities, we then obtain $u_1 + u_2 + \cdots + u_j \geq (1-\epsilon)OPT$.

Therefore, we conclude that AGQA algorithm achieves an approximation factor of $min(1/2, 1-\epsilon)$ for the special case of video quality assignment problem where $(3-2\epsilon)\epsilon \leq 1/S$.

REFERENCES

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

[2] Abdelhak Bentaleb, Ali C. Begen, and Roger Zimmermann. 2016. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In *Proceedings of the 2016 ACM Conference on Multimedia (MM'16)*. ACM, 1296–1305.

[3] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Commun. Surv. Tutor.* 21, 1 (2019), 562–585.

[4] BisectionMethod. 2019. Retrieved from https://en.wikipedia.org/wiki/Bisection_method.

[5] Kevin Boos, David Chu, and Eduardo Cuervo. 2016. FlashBack: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*. ACM, New York, NY, 291–304.

[6] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. 2012. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2.

[7] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. 2014. A hybrid edge-cloud architecture for reducing on-demand gaming latency. *Multimedia Syst.* 20, 5 (2014), 503–519.

[8] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. 2016. Design and experimental evaluation of network-assisted strategies for HTTP adaptive streaming. In *Proceedings of the 7th ACM International Conference on Multimedia Systems (MMSys'16)*. ACM, 1–12.

[9] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-degreee video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys'17)*. ACM, 199–204.

[10] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proceedings of the 2017 ACM on Multimedia Conference (MM'17)*. ACM, 943–951.

[11] Mohammed S. Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. 2018. Toward low-latency and ultra-reliable virtual reality. *IEEE Netw.* 32, 2 (2018), 78–84.

[12] Melike Erol-Kantarci and Sukhmani Sukhmani. 2018. Caching and computing at the edge for mobile augmented reality and virtual reality (AR/VR) in 5G. In *Ad Hoc Networks*. Springer, 169–177.

[13] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computing—A key technology towards 5G. *ETSI White Paper* 11, 11 (2015), 1–16.

[14] T-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'14)*. ACM, 187–198.

[15] Teemu Kämäräinen, Matti Siekkinen, Jukka Eerikäinen, and Antti Ylä-Jääski. 2018. CloudVR: Cloud accelerated interactive mobile virtual reality. In *Proceedings of the 26th ACM International Conference on Multimedia (MM'18)*. ACM, 1181–1189.

[16] Jonathan Kua, Grenville Armitage, and Philip Branch. 2017. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Commun. Surv. Tutor.* 19, 3 (2017), 1842–1866.

[17] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. 2017. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd International Conference on Mobile Computing and Networking (MobiCom'17)*. ACM.

[18] Zhu Li, Shuai Zhao, Deep Medhi, and Imed Bouazizi. 2016. Wireless video traffic bottleneck coordination with a DASH SAND framework. In *Proceedings of the IEEE Visual Communications and Image Processing*. IEEE Press, 1–4.

[19] LTEThroughput. 2009. Retrieved from http://www.etsi.org/deliver/etsi_tr/136900_136999/136942/08.02.00_60/tr_136942v080200p.pdf.

[20] Abbas Mehrabi, Matti Siekkinen, Gazi Illahi, and Antti Ylä-Jääski. 2019. D2D-enabled collaborative edge caching and processing with adaptive mobile video streaming. In *Proceedings of the 20th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE Press, 1–10.

[21] Abbas Mehrabi, Matti Siekkinen, and Antti Ylä-Jääski. 2018. QoE-traffic optimization through collaborative edge caching in adaptive mobile video streaming. *IEEE Access* 6 (2018), 52261–52276.

[22] Abbas Mehrabi, Matti Siekkinen, and Antti Ylä-Jääski. 2019. Edge computing assisted adaptive mobile video streaming. *IEEE Trans. Mobile Comput.* 18, 4 (2019), 787–800.

[23] OculusRift-Blog.com. [n.d.]. John Carmack's Delivers Some Home Truths On Latency. Retrieved from http://oculusrift-blog.com/john-carmacks-message-of-latency/.

[24] Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latre, and Filip De Turk. 2015. QoE driven rate adaptation heuristic for fair adaptive video streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 12, 2 (2015), 1–15.

[25] Stefano Petrangeli, Jeroen Van Der Hooft, Tim Wauters, and Filip De Turck. 2018. Quality of experience-centric management of adaptive video streaming services: Status and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 2 (2018), 1–29.

[26] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular (ATC'16)*. ACM, 1–6.

[27] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hossfeld, and Phuoc Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Commun. Surv. Tutor.* 17, 1 (2015), 469–492.

[28] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. 2019. Mobile VR on edge cloud: A latency-driven design. In *Proceedings of the 10th ACM Multimedia Systems Conference*. ACM, 222–231.

[29] Shu Shi, Varun Gupta, and Rittwik Jana. 2019. Freedom: Fast recovery enhanced VR delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'19)*. ACM, 130–141.

[30] Shu Shi and Cheng-Hsin Hsu. 2015. A survey of interactive remote rendering systems. *ACM Comput. Surv.* 47, 4 (2015), 1–29.

[31] SimuLTE. 2015. Retrieved from http://simulte.com.

[32] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal adaptation for online videos. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM'16)*. IEEE, 1–9.

[33] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM Conference on SIGCOMM (SIGCOMM'16)*. ACM, 272–285.

[34] Tuyen X. Tran and Dario Pompili. 2019. Adaptive bitrate video caching and processing in mobile-edge computing networks. *IEEE Trans. Mobile Comput.* 18, 9 (2019), 1965–1978.

[35] Cong Wang, Amr Rizk, and Michael Zink. 2016. SQUAD: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP. In *Proceedings of the 7th ACM International Conference on Multimedia Systems (MMSys'16)*. 1–12.

[36] Desheng Wang, Yanrong Peng, Xiaoqiang Ma, Wenting Ding, Hongbo Jiang, Fei Chen, and Jiangchuan Liu. 2019. Adaptive wireless video streaming based on edge computing: Opportunities and approaches. *IEEE Trans. Serv. Comput.* 12, 5 (2019), 685–697.

[37] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys'17)*. ACM, 193–198.

[38] Junfeng Xie, Renchao Xie, Tao Huang, Jiang Liu, and Yunjie Liu. 2017. Energy-efficient cache resource allocation and QoE optimization for HHTP adaptive bit rate streaming over cellular networks. In *Proceedings of the IEEE International Conference on Communication (ICC'17)*. IEEE, 1–6.

Received April 2020; revised October 2020; accepted October 2020