UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Offloading Strategies for the Rendering Process in Edge-enabled Virtual Reality Games

Cathy Schmit

September 27, 2022
Final project CS (192199978)

Design and Analysis of
Communication Systems (DACS)
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
The Netherlands

Abstract

For the optimal quality of experience (QoE), the user should play the VR game at the highest resolution, without stalls or quality switches and at a minimal start-up delay. However, current Head-Mounted Displays (HMDs) have insufficient local processing power to meet these requirements, creating the need for offloading solutions. When offloading, the HMD communicates computationally heavy tasks such as rendering a frame to an entity with more extensive computing capabilities and awaits the results. Multi-access Edge Computing (MEC) is an emerging computing paradigm that offers processing help at the cellular edge by dedicated servers at the base station. Due to the proximity to the user, MEC servers can render frames without introducing long round-trip delays. Aided by their considerable computing power, they can render frames significantly quicker than the HMD, such that offloading can reduce the end-to-end rendering delay. However, if channel conditions are adverse, the network transmissions endanger the latency requirement of VR applications that expect a response in milliseconds. Hence the decision on whether to offload or not must not be taken lightly. These rendering strategies that decide, per frame, the location of processing and the resolution quality are central in this work. The contributions of this thesis in relation to existing work are the focus on explainable rendering strategies and the holistic point of view on VR games. Instead of approaching the process on a frame-by-frame basis, the user's QoE is evaluated as a whole by considering all the aforementioned QoE metrics. For this purpose, a framework is developed that simultaneously simulates the video game playback at a fixed framerate and the rendering of frames based on the strategy's decisions. In addition, the effect of buffering is investigated. Rendering strategies designed around traceable decision-making guarantee the explainability of their performance. Results show that rendering strategies that frequently offload to the MEC servers *and* use encoding and decoding schemes achieve better QoE-score than those that render locally or do not use encoding. Furthermore, results show that dynamic strategies can inadvertently deteriorate the QoE despite higher data rates when frequently alternating between different resolutions. This result stresses the importance of only rendering frames at higher resolutions when able to sustain this for an extended period.

Contents

List of Algorithms

List of Tables

List of Figures

Table 1: Table of symbols in the document.

Symbol	Name
h	Compression ratio used in [4].
c	Offloading percentage used in [4].
i	Index of a specific frame.
α	Number of users at the base station.
Z_C	Total processing power of the MECs.
z_c	MECs' processing power allocated to the user.
B	Bandwidth.
B^u, B^d	Up- or Downlink bandwidth.
γ	Signal-to-noise-ratio in decibel.
γ_l	Signal-to-noise-ratio in Equation (4.1).
γ^u	Signal-to-noise-and-interference ratio (SINR) during uplink transmission (dB).
γ^d	Signal-to-noise-and-interference ratio (SINR) during downlink transmission (dB).
R	Data rate.
R^u, R^d	Up- or downlink data rate.
$l \times w$	Tiling scheme: frame divided into $l \times w$ tiles in length and width.
p	Pixels, unit of the resolution.
β	Proportion of a frame corresponding to the background.
D_t	Data size of tile.
D_f	Data size of foreground of a tile.
D_b	Data sizee of background of a tile.
D_u	Data size of frame needed for uplink transmission.
C_t	Computational requirement when rendering tile.
C_f	Computational requirement when rendering foreground of tile.
C_b	Computational requirement when rendering background of tile.
D	Data size of frame.
C	Computational requirement when rendering frame.
D^i	Data size of frame i.
C_i	Computational requirement when rendering frame i.
D_k	Data size of encoded frame.
C_k	Computational requirement when decoding an encoded frame.
k	Compression ratio of encoding algorithm.
$\mathcal{O}$	Sequence of mode selection indicator, the mode indicates the offloading path to take to render the frame. Sequence means for every frame.
$\mathcal{Q}$	Sequence of quality selection, so the bitrate decision for every frame.
ϕ	Offloading strategy for every frame, $\phi(i) = (\mathcal{O}(i), \mathcal{Q}(i)), \forall$ frame i.
o	offloading decision, $o \in \{0, 1, 2, 3\}$.
q	Resolution of a frame in p(ixels), $q \in [360, 480, 720, 1080, 2160]$.
n	Number of tiles in the FoV.
z_l	Local processing power.
$t_{integrate}$	Delay to integrate back- and foreground of a frame on the HMD.
b_{z_c}, b_{z_l}	Number of bits processed in a CPU cycle at MECs or locally.
θ	Effective bitrate of VR game-play.
Θ	Maximally achievable bitrate of VR game-play.

T_0	Start-up delay before displaying frames.
T	Total play time.
T_S	Total stall duration, where HMD waits for ready frames.
$D_{switches}$	Bits involved in quality switches.
D_{played}	Total bits played.
b	Number of frames to buffer before displaying first frame.
η	Desired framerate for the playback of the VR game.
N_T	Number of frames played in total time T.
T_{local}	Local offloading delay.
T_{remote}	Remote offloading delay.
T_{coop}	Cooperative offloading delay.
$T_{remoteenc}$	Remote encoding offloading delay.
$\mathbb{I}$	Indicator function $\rightarrow \{0,1\}$.
F	Ordered set of all frames played.
κ_{1-4}	Coefficients of the QoE-score.
σ	Perturbation parameter of the GREEDY stragegy: by how much the actual SINR value is perturbed to simulate an estimation.
q^*	Default resolution to choose for the OPP-BUFFER strategies
t_{2160}, t_{1080}, t_q	Delay threshold to choose the quality 2160p or 1080p or qp for the OPP-BUFFER strategies.
$O(n_q)$	Complexity of the GREEDY and OPP-BUFFER heuristics that choose out of n_q quality resolutions.
M_B	Maximum buffer size.
M_α	Maximal number of users that can be accommodated at the base station such that the offloading delay is below $1/\eta$.
$U[a,b]$	Uniformly distributed between a and b.
θ_t	Threshold for the QoE-utility defined by Guo et al. [16].
T^{local}	Local rendering delay as computed by Guo et al. [16].
T^{remote}	Remote rendering delay as computed by Guo et al. [16].
T^{coop}	Cooperative rendering delay as computed by Guo et al. [16].
T_q	Expected rendering delay when rendering at a resolution q.
T_o^i	The (expected) rendering delay of frame i using the offloading path o
κ'_{1-4}	Coefficients of the objective function that the heuristic that more directly intends to optimise the QoE score uses.

Table 2: Abbreviations in the document

Abbr.	Name
AR	Augmented Reality
BS	Base Station
DASH	Dynamic Adaptive Streaming over HTTP
E2E	End-to-end (delay)
FoV	Field of View
FPS	Frames per second
HMD	Head-Mounted Display
MEC	Multi-Access Edge Computing
MECs	MEC server(s)
UHD	Ultra-high definition
VR	Virtual Reality

Chapter 1

Introduction

Virtual reality (VR) games are becoming increasingly popular, not only in entertainment but also in education and healthcare [9, 46], because of the immersive experience it provides. However, some technical challenges must be overcome to secure its spot in consumer electronics. VR videos and especially VR games (requiring real-time rendering) have ambitious requirements concerning latency, computation, throughput and energy consumption. For a satisfying user experience, the VR headset must display a game's frames at a sufficiently high framerate and resolution without interruptions or "stalls". Often, this computational burden exceeds the VR headset's processing capability. Hence, the Head-Mounted Display (HMD) can offload the computations of the rendering process to a more powerful device to save time and energy. While cloud servers have immense processing power, the high round-trip time to communicate information makes them infeasible for an application that needs a response in milliseconds. Therefore, Multi-Access Edge Computing (MEC) is a promising technology to satisfy these demands. Located at the base station, MEC servers bring computational resources closer to end devices and can help with their computational burden without introducing considerable round-trip delays. Whether to render a frame locally at the VR device or remotely at the edge is called the 'offloading decision'. The challenge in making this decision is balancing the high quality and low latency demand because rendering frames at higher resolutions takes longer and requires more bandwidth during the network transmission.

To reduce the bitrates of VR applications, video streams and games make use of *tiling* and *bitrate adaption schemes* [17]. As a human's field of view is smaller than the 360° offered by a HMD, the video size can be reduced by dividing the video spatially into tiles and only rendering the tiles lying in the user's field of view (FoV) at a higher quality.

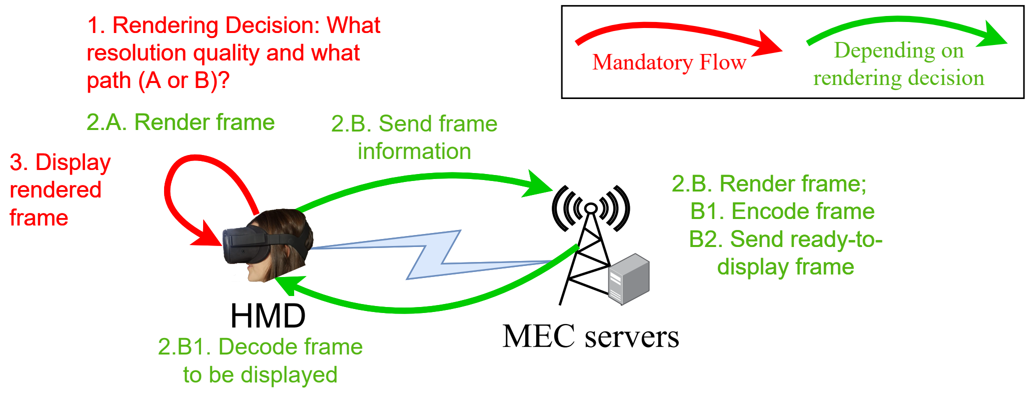

Mandatory Flow
Depending on rendering decision
1. Rendering Decision: What resolution quality and what path (A or B)?
2.A. Render frame
2.B. Send frame information
3. Display rendered frame
2.B. Render frame;
B1. Encode frame
B2. Send ready-to-display frame
HMD
2.B1. Decode frame to be displayed
MEC servers

Figure 1.1: Overview of the system. The HMD can offload the rendering of frames to the MEC servers.

In recent years, literature on MEC-enabled offloading has vastly increased. On *scopus*, Elsevier's "abstract and citation database " [12], has marked an increase of entries using "MEC-enabled offloading"-like keywords from 31 in 2016 to 1004 in 2021. MEC-offloading algorithms that use machine-learning techniques follow a similar trend (Appendix A provides the statistics in more detail). Also, many state-of-the-art algorithms in VR attempt to utilise the strengths of multi-access edge computing. [28, 4]. The key motivations for this thesis are the following observations:

- Due to the many factors involved in offloading and the complexity of algorithms, it is difficult to explain the exact benefit of an algorithm and evaluate the impact on the system environment.
- Rendering strategies for VR games generally evaluate an algorithm's performance on how individual frames are rendered instead of evaluating the quality of experience as a whole [28, 16], as done by VR streaming algorithms [20, 4, 17]. However, streaming algorithms cannot be directly applied to VR games because they assume video encoding happens in advance, and VR games require real-time processing.

Based on these two motivations, we first aim to find explanations for the performance of a state-of-the-art rendering strategy for VR games. Then, we use the gained insights to devise simple and *explainable* rendering heuristics and evaluate them from a holistic point of view that incorporates the following streaming aspect. The HMD displays frames at a fixed framerate; if they are not ready in time, a stall occurs, which deteriorates the user experience. This incorporation is the main contribution of this thesis compared to existing work and intrinsic to its positioning: devising offloading decisions for rendering VR games in a MEC-enabled environment under 5G and beyond data rates while considering streaming aspects. The remainder of this chapter first clarifies the above concepts, then presents the research questions and lastly provides an overview of the organisation of the thesis.

Figure 1.1 shows the overview of the system flow. When playing the game, the two key responsibilities of the HMD are rendering frames and displaying them. The rendering task is either entirely computed *locally* (Computation path 2.A in Figure 1.1) or by *offloading* (parts) of the rendering task to the MEC servers (path 2.B). The rendering decision consists of two parts: choosing the resolution quality (called bitrate decision) and choosing the offloading path (called offloading decision). In our VR gaming setting, the possible offloading paths are:

0. **Local rendering.** The entirety of the next frame is rendered locally at the HMD, so it produces all the game content by itself. No connection to the base station is necessary.
1. **Remote rendering.** The MEC servers render the entire frame. To do so, the HMD first communicates user input (in-game actions, expected FoV) to the MEC server that determines how to render the next frame. Then, after the MEC servers have produced the game content of the frame, the HMD downloads the raw frame over the cellular link and displays it to the user.
2. **Cooperative rendering.** The foreground activity in the frame is rendered locally, while the background information is communicated to the MECs and rendered remotely. In the end, the HMD integrates both images into the final frame displayed to the user.
3. **Remote rendering with encoding.** The procedure is equal to remote rendering except that the frame is encoded (compressed) before the downlink transmission and decoded again at the HMD. This reduces the downlink transmission delay by reducing the data needed to transmit over the network, but it requires additional encoding and decoding computations.

We define the *rendering delay* as the end-to-end delay between starting the rendering process of a frame and putting the ready-to-be-displayed frame into the buffer of the HMD. Moreover, a *dynamic* rendering strategy can have a different decision for every frame, while a *static* rendering strategy always chooses the same offloading path and resolution quality.

The VR game is played at a fixed framerate of 60 frames per second (FPS) unless there is no frame left in the buffer. Then, the system *stalls* until the new frame is rendered. When stalling, the game stutters or even freezes, significantly reducing the player's quality of experience (QoE). To prevent this, frames can be rendered ahead and stored in the buffer until needed. However, in VR games, the content of the video changes based on user actions, so rendering cannot be done entirely in advance but has to happen during the play-through. In this work, we assume that all information necessary to render a frame can be accurately predicted a short time ahead (at most 0.5s), allowing us to study the impact of a buffer system.

In addition to reducing the risk of stalls, the HMD aims to maximise the video game's resolution quality, minimise quality changes between frames, and reduce the start-up delay. All these components influence a player's QoE and must be balanced. For example, rendering frames at higher resolutions increases the video quality, but it takes longer, increasing the risk of a stall. Another example is rendering frames at higher resolutions. If the buffer is filled, this increases the quality of the game. However, it might come at the expense of many quality switches or stalls when the buffer is depleting. The goal of the HMD is to have a rendering strategy (making a rendering decision at every frame) that optimises the overall QoE of a user. To capture these four aspects in a single, measurable score, we define the *QoE-score* after the principles of existing QoE-models [39, 40] to evaluate the performance of rendering strategies.

Now, we can formulate our research questions:

1. Do dynamic rendering strategies outperform static rendering strategies in terms of QoE-score? To stress the importance of explainable strategies, we focus on strategies with an "if-then" character that allow us to investigate why a dynamic strategy might be superior to its static counterpart:
   (a) Do dynamic strategies that consider current channel conditions achieve high QoE-Scores?
   (b) Do dynamic strategies that consider the buffer space achieve high QoE-scores?
   (c) What offloading path (local, remote, coop and remote rendering) leads to high QoE-scores?
2. What is the impact of system parameters on the QoE-score of different rendering strategies? Concretely, we evaluate the impact of the following system parameters:
   (a) What is the impact of the up- and downlink bandwidth on the QoE-score of rendering strategies?
   (b) What is the impact of the maximum buffer capacity on the QoE-score of rendering strategies?
   (c) What is the impact of the channel quality on the QoE-score of rendering strategies?
   (d) What is the impact of the compression ratio on the QoE-score of strategies that use remote rendering with encoding?

Answering the above questions analytically is difficult, because the rendering decision of later frames depends not only on the rendering decision of earlier frames but also on the experienced channel quality which influences the number of frames that might already be buffered. Moreover, the rendering and playback processes happen simultaneously, which is difficult to model mathematically, so we resort to simulations. To our knowledge, there is no dedicated holistic framework that simulates the rendering process of a VR game and evaluates the different components of the QoE-score, so we design and propose our own simulation framework.

To answer RQ 1, we design two types of rendering heuristics (in line with RQ 1a and 1b, respectively) and compare them to strategies with static decisions. The first type of strategy, called GREEDY strategies, considers (estimates of) the current system state, such as the processing capability of the HMD and the MEC servers, the available bandwidth and the experienced channel quality. The second type of strategy, named OPP-BUFFER strategies, takes into account the current state of the buffer and renders at higher resolutions if enough frames are buffered already. Both strategies have a primary goal to reduce the risk of a stall and a secondary goal to maximise the resolution quality of frames. To answer RQ1c, we devise different variants per heuristic that favour a specific offloading path. To respond to RQ 2, we explore the parameter space by simulating and evaluating the above heuristics over a range of different values for the bandwidth, buffer capacity, channel quality and compression ratio.

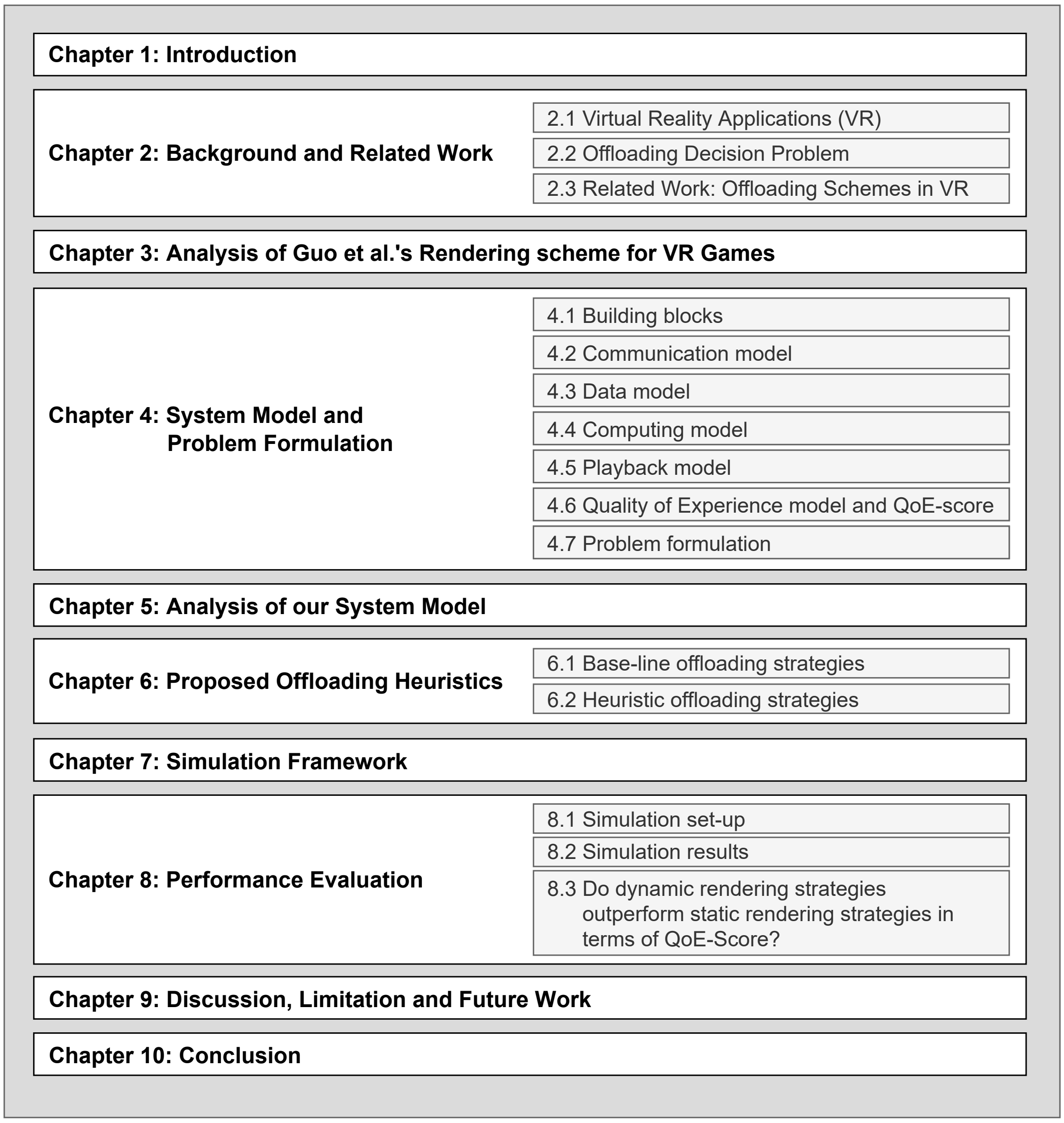

Chapter 1: Introduction
Chapter 2: Background and Related Work
2.1 Virtual Reality Applications (VR)
2.2 Offloading Decision Problem
2.3 Related Work: Offloading Schemes in VR
Chapter 3: Analysis of Guo et al.'s Rendering scheme for VR Games
Chapter 4: System Model and Problem Formulation
4.1 Building blocks
4.2 Communication model
4.3 Data model
4.4 Computing model
4.5 Playback model
4.6 Quality of Experience model and QoE-score
4.7 Problem formulation
Chapter 5: Analysis of our System Model
Chapter 6: Proposed Offloading Heuristics
6.1 Base-line offloading strategies
6.2 Heuristic offloading strategies
Chapter 7: Simulation Framework
Chapter 8: Performance Evaluation
8.1 Simulation set-up
8.2 Simulation results
8.3 Do dynamic rendering strategies outperform static rendering strategies in terms of QoE-Score?
Chapter 9: Discussion, Limitation and Future Work
Chapter 10: Conclusion

Figure 1.2: Structure of the thesis.

Figure 1.2 provides an overview of the thesis structure. After this introduction, Chapter 2 gives more detailed explanations of the background and related work regarding virtual reality (VR) and the offloading decision problem. It finishes by comparing and positioning our work against related work on VR streaming and gaming. Next, we analyse the performance of one of the state-of-the-art rendering algorithms in Chapter 3. This chapter aims to identify bottlenecks and explain the performance results of Guo et al.'s rendering scheme [16]. These insights are fundamental for formulating the system model in Chapter 4. Due to the many factors involved in the rendering process, this chapter explains how the different aspects are modelled and motivates the relevant assumptions and simplifications. The chapter finishes with the mathematical formulation of the underlying optimisation problem. Consequently, Chapter 5 analyses the offloading paths in our model under different parameters to determine their feasibility, similarly to the analysis of a state-of-the-art algorithm in the earlier chapter. Afterwards, these insights are used to design the heuristics GREEDY and OPP-BUFFER presented in Chapter 6. This chapter explains the underlying idea of the two heuristics in more detail, highlights their advantages and disadvantages, and presents the static rendering strategies. Chapter 7 introduces the design and functioning of the simulation framework, and Chapter 8 presents the performance evaluation in light of research questions 1 and 2. The thesis is wrapped up by Chapters 9 and 10 that present the discussion, limitations, future work and conclusion.

Chapter 2

Background and Related Work

This chapter provides detailed information, first, about the different aspects of Virtual Reality (VR) like the Field of View (FoV) and tiling schemes (in Section2.1), then, more generally, about the offloading decision problem (in Section 2.2). The chapter concludes by zooming into existing offloading schemes in literature and by positioning our work in Section2.3.

2.1 Virtual reality applications (VR)

Compared to videos and games experienced via a flat screen, such as a television screen or a monitor, applications in virtual reality completely immerse the user in the digital environment. By wearing a Head-Mounted Display (HMD) as in Figure 2.1, the user no longer sees any objects in the actual room because the digitally displayed image takes over the entire vision. VR videos and games display inherently 3D pictures, either recorded by video cameras as in most 360° videos or by rendering 3D models as in many VR games [3]. Moreover, the individual VR images span the entire 360° range allowing the user to turn their head and take in new views, just like in reality.

Using motion sensors to track head movements such as rotation, tilting and scaling, the HMD can calculate which parts of the picture lie in the user's field of view (FoV). The field of view is defined as the vision range that the user can perceive at once, as visualised in Figure 2.2. In literature, this is also called *viewport* [17]. According to [14], humans have a field of view of around 110 degrees, so we can calculate the proportion of the image that the user can see.

Figure 2.1: Example of a VR head-mounted display with controllers: Oculus Rift [8]

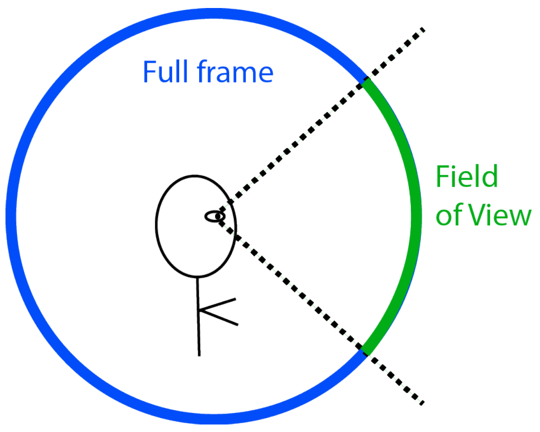


Figure 2.2: Illustration on a user's Field of View (FoV).

For a VR game, new images must be generated based on mathematical structures, called *rendering*. Much research studies efficient rendering models that produce more realistic audio and graphics [3, 44]. As video games are highly interactive, they need to render new images during the gameplay to account for a user's actions.

For a VR video, the editor joins individual recordings, audio and images into one final 360°video. After the raw video is finalised, it is compressed by a video encoder if it has to be transmitted over the network. These steps are usually done before watching the video. To watch the video, the HMD requests the content, receives the compressed files and decodes them into a format that can be displayed. Contrarily, rendering must be done on the spot for a VR game and does not necessarily require a server connection.

Since a video consists of a sequence of images that are displayed in quick succession, the developer's goal is to provide a stream of images at a sufficiently high *framerate* and *resolution quality* to ensure an excellent *quality of experience* (QoE). The *framerate* corresponds to how often the displayed screen is refreshed: at 60 frames per second (fps), every $1/60 \sim 0.167$ s, the user gets a new image on the screen [37]. Suppose a system performance cannot reach a high enough framerate. In that case, the video experience can be "choppy": instead of a smooth video, the individual pictures are hopping from one to another, which is generally an unpleasant experience. The more action (movement, crowded scenery or the viewing speed), the more critical a high framerate is. For example, when looking at a sunset, lower framerates are acceptable since the individual pictures do not differ significantly, but watching a fast-paced race or fighting series requires higher framerates for a smooth experience. Most researchers aim for a framerate of 60 FPS [15].

When the framerate is fixed, and the HMD cannot produce the required frames per second, the system enters a *stall*. This means the streaming buffer (the queue of ready-to-display frames) is empty before the newly rendered frame arrives. Then, the user's screen abruptly stops (often accompanied by the dreaded loading sign), causing frustration. Hence, *stalls* should be avoided at all costs. Especially since multiple short stalling periods or a few long stalling periods are suspected of causing cybersickness [42]. The latter refers to the phenomenon that users experience dizziness or even nausea after the VR experience.

A third metric related to the framerate inherent to VR games is the *response time*. It measures how long it takes between the *action registration* and the *action consequence* being shown on screen. As explained in [37], so-called *double buffering* is commonly employed when playing VR videos: to ensure smooth video, a frame, say frame t, is shown on the screen while the next frame $t+1$ is already prepared and stored in the buffer. To react to user actions, the first frame that can show the adaptions is then

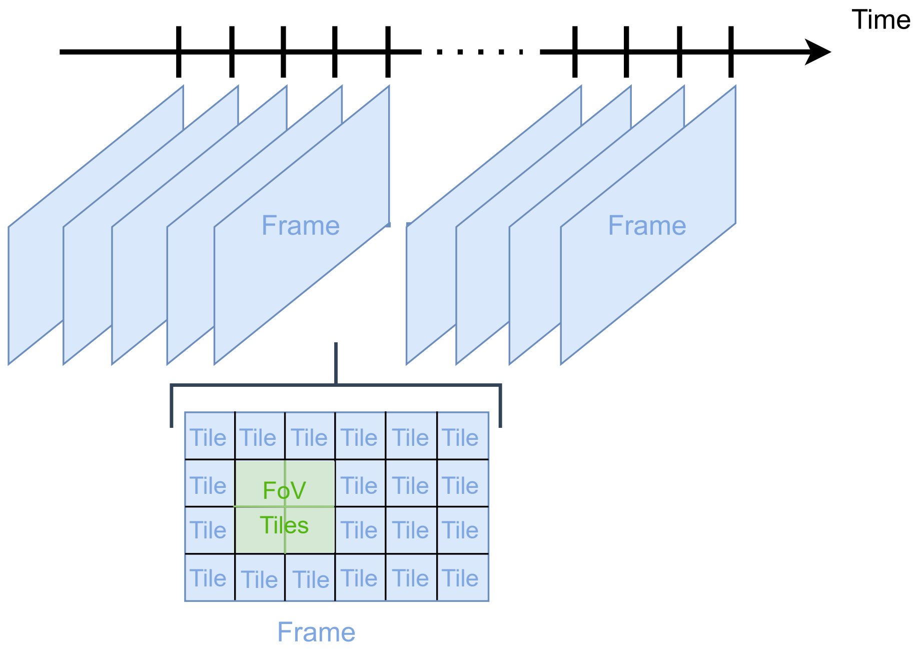


Figure 2.3: Illustration on how a VR video is time-wise split into frames and spatially into tiles of which part belong to the field of view (FoV).

frame $t+2$, which is prepared (fetched, rendered and decoded.) while frame $t+1$ is playing. Ergo, the minimal response time is one frame (or 16.7ms), but if the processing related to the user action takes longer, the changes will only be visible at a later frame. The longer this response delay, the worse the user experience, commonly known as "laggy behaviour".

Other primary influences on the quality of experience are the *resolution quality* of the individual images and how the quality *switches* between frames. Both too many quality switches and a low resolution deteriorate the viewing experience [7].

The main challenge in VR videos and games is to balance the above factors because rendering and downloading a video at a higher resolution takes significantly longer, so the risk for stalling periods increases. Images of higher qualities also result in higher file sizes causing longer transmission delays. To combat the latter and use the VR-specific property that a user only sees the parts of the image lying in their FoV, adaptive bitstream algorithms make use of tiling [17]. On top of splitting the video time-wise into frames, every frame is spatially split into *tiles* as shown in Figure 2.3. Every tile can then be rendered at a different resolution allowing the possibility to render the FoV at a higher quality than the rest of the image. In this way, the user's quality of experience remains unchanged, but the file size is significantly reduced. A challenge of adaptive tiling schemes is correctly predicting the user's FoV and then rendering the appropriate tiles [17].

2.2 Offloading decision problem

In short, computational offloading is a part of resource management: how to effectively share computational resources among several, possibly heterogeneous, devices to optimise performance metrics such as energy consumption and latency by *offloading* (i.e. transferring) some tasks from one entity to another. By offloading computationally heavy tasks to entities with more considerable processing capabilities, battery-powered end devices could save energy and shorten the end-to-end delay. However, they have to engage in an extra transmission with all its potential costs, such as round-trip delays, energy costs related to the transmission power, potential jitter and packet loss and extra traffic in the network.

Context and incentives to offload

Emerging and future applications such as VR or autonomous driving require the processing of large quantities of data in a very short time, to the extent that the hardware present on a mobile device might not be capable of handling the request by itself [43]. Emerging (ultra) high-speed network technologies such as Wi-Fi and (beyond) 5G can support large data transfers in a relatively short time [41]. Therefore, it is possible and even beneficial to not execute all computations in a local device but to distribute (offload) computing and let more powerful devices such as cloud servers handle heavy computations. After the processing, the MECs transmit the results back to the local device. However, the network infrastructure needs to bring computational power closer to the end-user to offer ultra-low latency. This led to the advancements of multi-access edge computing (MEC): processing capabilities and storage, albeit less potent than those of the cloud, are made available at the *edge* of the cellular network. In short, the main advantages of computational offloading for an end-user consist of:

- Improving end-to-end latency. The lower the processing power of a device, the longer computations take. Therefore, offloading certain tasks to destinations with better processing power can reduce end-to-end latency despite the additional communication delay.
- Improving battery life. By offloading computationally heavy tasks whose processing requires significant energy to another destination, end devices could consume less energy and have a higher battery life span despite the additional energy consumption to transmit the data.

At the uprising of computational offloading, cloud centres were the most prominent destination because of their enormous processing capabilities. However, their shortcomings and emerging technologies lead to a new popular destination, the mobile edge. Due to the proximity of the edge to the end devices, offloading to the mobile edge can alleviate the shortcomings of cloud-based offloading:

- Shorter delays. Since cloud data centres are typically located far from the end device, offloading to them introduces considerable round-trip delays. Therefore, it often only makes sense to offload highly computationally intensive tasks to the cloud, whose processing time will be significantly shorter in the cloud than at the end device. In contrast, multi-access edge computing happens at the base station, so significantly closer to end devices and introduces a lower round-trip delay.
- Reduced load on core infrastructure. Offloading to the cloud uses the core internet infrastructure to communicate potentially sizeable input data for a task and send the response back. Offloading to the edge avoids this additional network traffic due to its proximity to end-users.
- Contextual information. When end-users are connected to the cellular network, the cellular edge can directly store and access contextual information related to end-users, such as their location and experienced link quality [11], without requiring additional bandwidth resources. The cloud does not have access to this information by itself. Therefore, if the information is required, it must be explicitly communicated, increasing the amount of data the end device needs to send. Furthermore, edge servers can have access to the same environmental information as end devices. For example, an edge server located next to monitoring sensors could also measure the current temperature and relieve the sensors of this burden.

Factors that play a role in offloading

The classification in Figure 2.4 categorises the different factors that influence the modelling of the offloading decision problem. We distinguish the following aspects: the (physical) *system* is the foundation of what is possible. The top-layer *application* tasks use the system and pose requirements for proper functioning. The offloading *decision problem* combines needs and capabilities to achieve a common goal. Appendix B explains all the factors of Figure 2.4 in more detail.

Additional design options to consider when offloading in VR

Compared to offloading decisions in other application areas, the VR context obliges us also to consider the following aspects.

- **Buffer size (System design choice).** When designing *buffer space*, the risk of stalls needs to be jointly considered with spacing constraints and response delay. Intuitively, the larger the buffer size, the lower the risk of stalling. However, the main disadvantages of large buffers are increased memory usage and a potentially increased response delay. If the frame with the user response is added at the end of the buffer queue, playing all older frames in the buffer first can significantly increase the experienced response delay. On the other hand, if the same frame overwrites an older, outdated frame, computational resources have been wasted since rendering the old frame was unnecessary. Hence, for the optimal maximal buffer size, the risk of stalling, response time and memory need to be in equilibrium.

- **Choosing video quality (Strategy for video encoder).** The higher the video quality, the better the user experience. However, the associated delays can reduce the experience when response time, framerate, or stalling metrics deteriorate. To balance this out, *bitrate adaption schemes* like DASH are widely used in video streaming: DASH segments the video preliminarily and encodes each segment into different quality streams. At the moment of request, it chooses the new segment's quality dynamically depending on the channel quality to optimise the performance [17]. These segments can range from single frame segments over multiple frame segments to not using DASH (the whole video is one segment).

  During VR videos and games, this idea is taken a step further. Instead of only splitting the video time-wise into segments of differing quality, each frame is also segmented spatially. This is possible because the user only sees the parts of the frame in their FoV. Dynamic DASH for VR divides every frame into a grid of tiles [17], each tile encoded into different qualities. Depending on the user's FoV, only the tiles with parts in the FoV are rendered in high quality, while all tiles outside are rendered in low quality not to waste bandwidth and processing power. Since tiles outside the view are still rendered, stalls are prevented in case the user makes a rash movement towards the tile.

- **VR game versus 360-degree video** The conceptual difference between a VR game and a 360° video is as follows. A VR video is an immersive 360° video capture of a real environment. All-around video captures can be recorded, stored and encoded to different qualities. The only user action possible when watching such a video is rotating and tilting one's head to view different angles of the video. Moreover, the video always displays the same series of images. On the contrary, a VR game is more flexible. While it can be based on real-life imagery, a game is commonly played in a digital VR environment that is rendered on the spot. Notably, the user has a lot more freedom in the interaction with this virtual world: on top of head motions to see different angles, the user can move around and interact with virtual elements. Since it is unrealistic to anticipate the user's every move and action, the frames need to be rendered on the

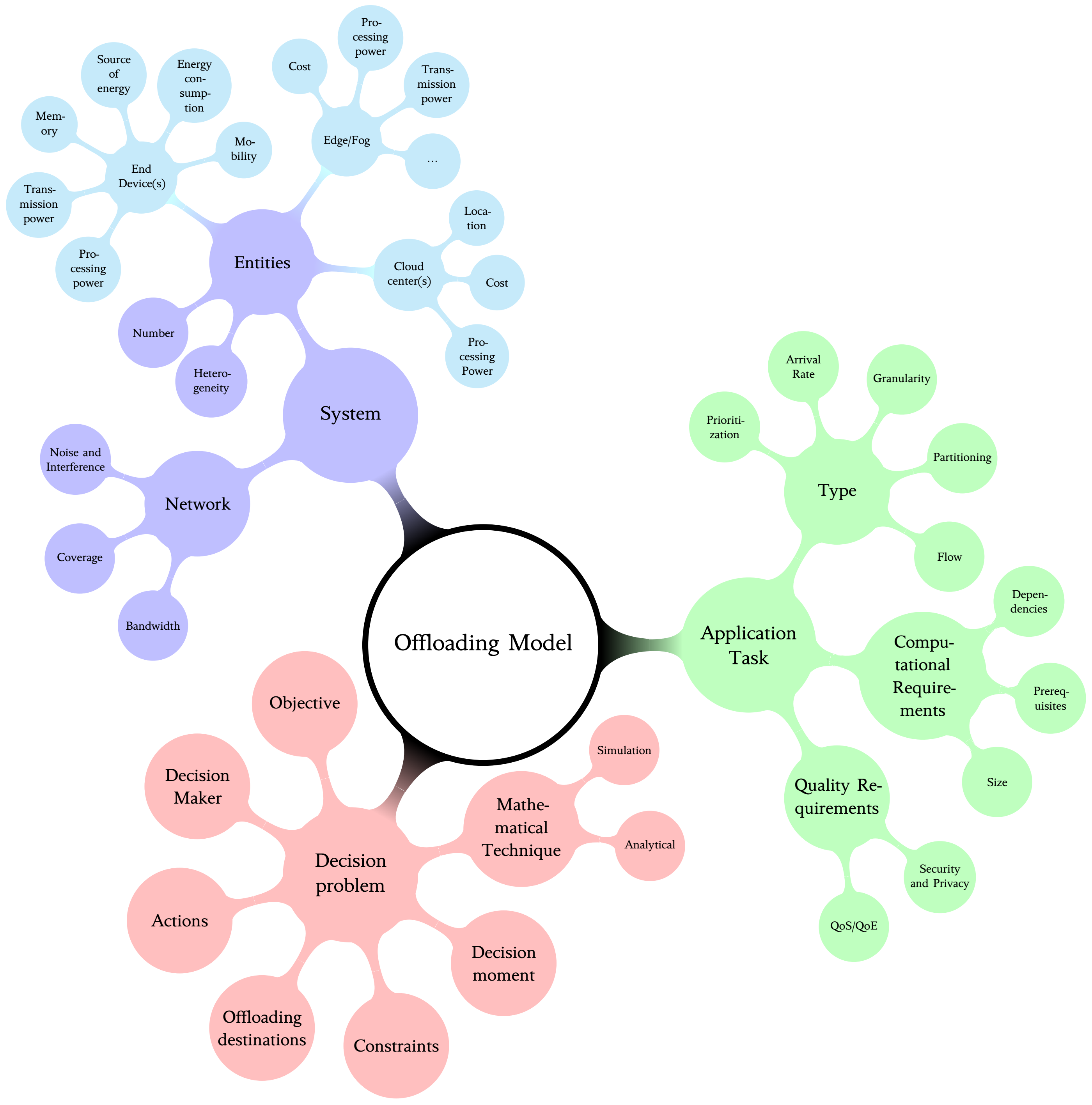
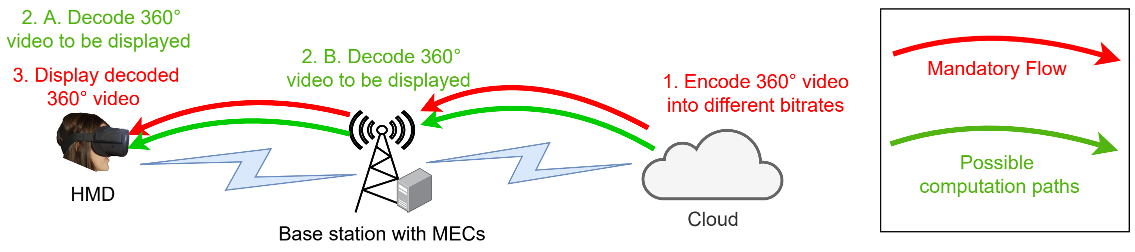
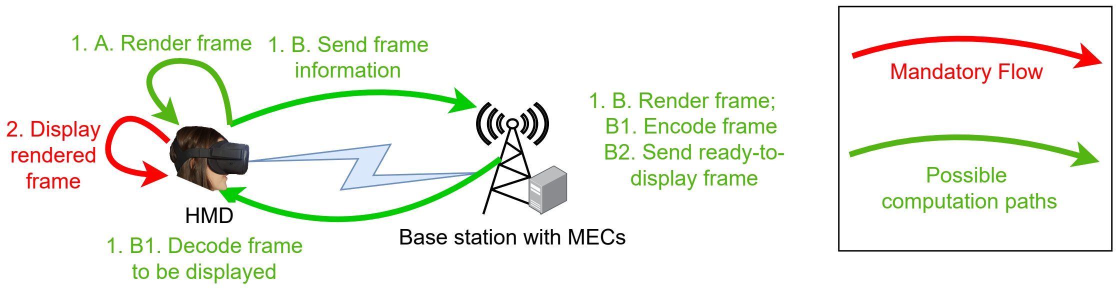


Figure 2.4: Classification of factors involved in modelling the offloading decision problem.


2. A. Decode 360° video to be displayed
3. Display decoded 360° video
2. B. Decode 360° video to be displayed
1. Encode 360° video into different bitrates
HMD
Base station with MECs
Cloud
Mandatory Flow
Possible computation paths

(a) VR Game.


1. A. Render frame
1. B. Send frame information
2. Display rendered frame
1. B. Render frame; B1. Encode frame B2. Send ready-to-display frame
HMD
Base station with MECs
1. B1. Decode frame to be displayed
Mandatory Flow
Possible computation paths

(b) 360° streaming.

Figure 2.5: Mandatory flows and possible computation paths of a system hosting a VR gaming or streaming a 360° video application.

spot to display the result of the user input. The available moves and actions can highly depend on the games.

From a technical and modelling perspective, we must highlight the following key differences that play a role in this research. The first key difference between both applications for our set-up is depicted in Figure 2.5, showing the mandatory steps in red and the different computing options in green. Figure 2.5a shows that *offloading* computations from the HMD to the MECs in a VR game introduces *new* transmissions. Contrarily, as depicted in Figure 2.5b, during VR *streaming*, the data has to flow from the content provider (cloud) over the base station to the HMD in any case. Then, the offloading decision is not *whether to offload the rendering to the MECs*, but *where to decode the compressed video files sent from the cloud*. Contrarily, in the VR gaming setting, offloading is entirely optional. So the trade-off between computations saved on the HMD and added network load is more significant. When remotely rendering, the system needs to account for an additional uplink and downlink transmission between the base station and HMD on top of the extra computations at the MECs. Moreover, the 'offloading decision' when streaming a 360° video can still be incorporated: we can consider remote rendering without compression (the raw file is sent over the network, no decoding needed at the HMD) or with compression (extra encoding delay at MECs, less volume on the downlink, but additional decoding action at the HMD).

The second key difference is how to deal with user interaction. When streaming a video, the 360° content is always the same, but users can see different parts of the video depending on their head movements. To ensure the FoV is displayed at the highest possible quality under given network conditions, the content provider can encode the video and even individual tiles at different quality resolutions [17]. Bitrate adaption schemes then determine in real-time which of the streams to transmit. The immutable video content permits encoding and caching of these data streams ahead of time, so no time is lost at the moment of request (computational delay is traded for memory). When playing a real-time VR game, the content of a tile is not known beforehand because it highly depends on the user's behaviour in the game. Ergo, the pre-rendering and caching options are limited. To conclude, we considered VR gaming a more interesting scenario to study the offloading decision problem.

2.3 Related work: offloading schemes in VR

This section presents the most relevant work on VR streaming and rendering leveraging edge and cloud infrastructure. Our work concentrates on rendering strategies for VR games, but we intend to combine the successful insights of existing streaming and rendering algorithms and mitigate their caveats.

Hooft et al. explore tile-based adaptive bitrate algorithms together with FoV prediction algorithms [17]. Their findings conclude that the viewport prediction accuracy can be improved by modelling the virtual movement of a user as a walk on a sphere instead of a two-dimensional representation on a plane. Moreover, they show that their tile-based rate adaption heuristics outperform state-of-the-art algorithms. Their work instigates us to assume the presence of accurate FoV prediction algorithms and to consider rendering schemes that adopt different quality resolutions for FoV and non-FoV tiles. We apply their design philosophies to a new context, namely VR gaming in a MEC-enabled environment instead of 360° video streaming from a content provider.

Cheng et al. design and analyse a proactive 360° VR video streaming model to investigate the benefit of caching. They also investigate offloading by determining what percentage c of the VR video should be decoded already at the MECs or only at the user. They conclude that it highly depends on the compression ratio h, so how much larger the file size is after decoding. When $h = 5$, the end-to-end (E2E) delay when decoding only 70% at the MECs is shorter than the E2E delay when decoding entirely at the MECs. However, it is the contrary when $h = 1.5$ [4]. Nonetheless, it is noteworthy to mention two of our own observations: first, they only present results for $c = 0.7, 0.8, 0.9, 1$. This implies that they see promise in decoding already at the MECs, which is an inspiration for our proposal of not employing encoding and decoding when rendering remotely in Section 4.4. Secondly, the E2E delay seems to hardly be below the target of 1/60s.

Li et al. have developed an analytical framework to analyse the offloading paths to determine the minimum transmission data rate needed for the different offloading paths to meet the latency requirements [21]. They consider a VR video streaming system, where individual frames need to be transformed from 2D images into 3D images. This transformation can either be computed locally, at the MEC servers or in the cloud. Moreover, they investigate the influence of caching. Their methodology is the main inspiration for the analytical analyses in Chapters 3 and 5. The main difference to their system model is that we assume a VR gaming setting where frames need to be rendered and cannot be cached. Furthermore, we consider the possibility that up- and downlink capacities can vary and that the rendering task can be shared between HMD and MECs (cooperative rendering).

Lai et al. evaluated several VR applications on the proclaimed "VR ready" PIXEL phone by Google- and concluded that today's (mobile) hardware cannot satisfy wireless VR needs and that QoE values are 10x worse than required. Nonetheless, they are pioneering the observation that fore- and background activity in a VR game differ and hence develop the VR framework Furion. The framework runs on both mobile device and server in a "split renderer architecture" [18], where the foreground is rendered at the mobile device and the background at the remote server. In this way, Furion manages to achieve the desired 60 FPS. In their setting, a VR game is played on a mobile phone connected to a local desktop via a WiFi connection. Our research intends to analyse the VR experience with more powerful hardware: HMDs with dedicated processors to display VR, MEC servers with higher processing power than a desktop and the promised Gbps data rates by 5G and beyond networks. Furthermore, they empirically measure the rendering capability of existing applications, while our goal is to design new rendering strategies that take into account current network conditions for offloading and bitrate decisions. The main similarities between ours and Lai et al.'s work are that both consider VR games and allow for cooperative rendering.

Mehrabi et al. evaluate edge versus cloud VR gaming by employing a multi-tier structure (HMD, edge and cloud). At every time period, the frame can be rendered either at the edge or in the cloud. However, their numeric results have to be taken with caution because they "exclude the processing delays associated with client device" [28] since it is irrelevant to the question of whether to render in the cloud or at the edge server. But our exploratory results in Chapter 3 will show that the delay associated with local computations is often limiting, so it should not be ignored in the bigger picture. Furthermore, they evaluate the rendering process on a frame-by-frame basis while we consider playing the game for an extended period of time and examine a user's overall quality of experience.

Guo et al. already started extending Lai et al.'s ideas by involving the edge network. They run the split-renderer architecture on a dedicated HMD combined with MEC servers [16]. However, they intend to follow the real-time rendering principle implying that the background is rendered at the same time as the foreground even when cooperating between HMD and MECs. The two main shortcomings of their work are that they consider low-quality resolutions and only evaluate the system on a frame-by-frame basis. The latter implies that they cannot measure the effect of a single long frame computation on the overall user experience. They only calculate how many frames meet the latency requirement but not the resulting effect on the QoE metrics such as stalling duration or quality switches. In addition, their rendering strategy is based on deep learning and game theory which complicates explaining what exactly happens in the system (which decisions are made and how system parameters impact the performance). We extend their work by designing a model and simulations that take into account this streaming-like aspect.

Da Costa Filho et al. proposed a model to evaluate the QoE based on the resolution quality, the stall duration, the quality switches and the start-up delay [7]. Other studies such as [40] and [39] also model their QoE model on the same factors. The QoE model presented in Section 4.6 is based on the same principles but makes use of normalisation to establish the optimum and allow for a better comparison between QoE metrics of different magnitudes (e.g. bitrates in MB/s and stall duration in s).

To conclude, Figure 2.6 summarises how our work is positioned relative to the above selection of state-of-the-art literature by showing the key differences to our work. The main difference between Li et al., Cheng et al., and Hooft et al. [21, 4, 17] is that we consider a VR gaming setting with optional offloading instead of a streaming application. The key difference to Furion is that we consider a MEC-enabled setting with optional offloading. Moreover, we incorporate the streaming-like aspect of a game (displaying frames at a fixed framerate) and the impact of local computations, as opposed to Guo et al. [16] and Mehrabi et al. [28]. Last but not least, we take into account buffering capabilities at the HMD for VR games by assuming that user actions and FoV can be predicted a short time ahead.

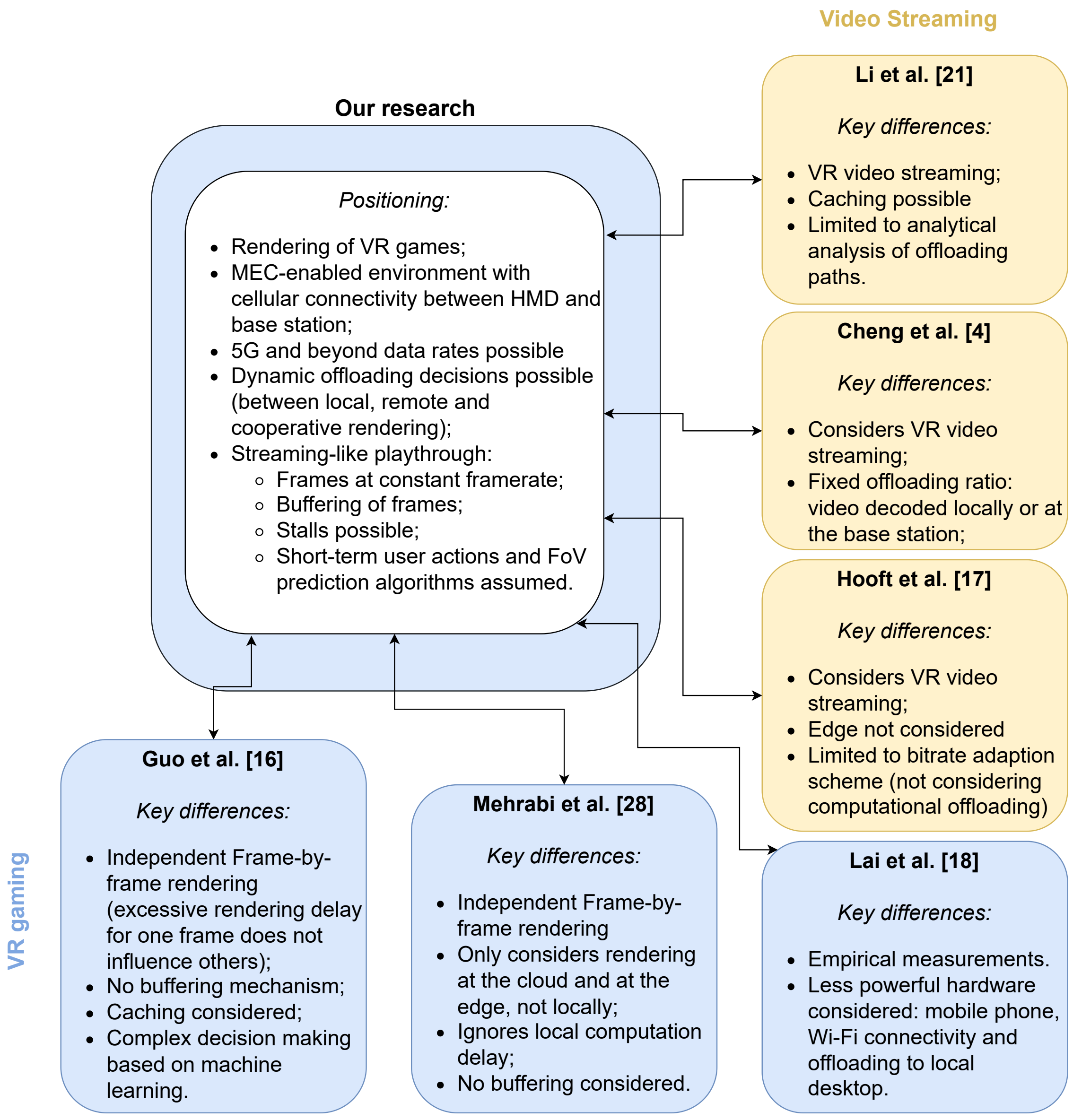

Video Streaming
Our research
Positioning:
Rendering of VR games;
MEC-enabled environment with cellular connectivity between HMD and base station;
5G and beyond data rates possible
Dynamic offloading decisions possible (between local, remote and cooperative rendering);
Streaming-like playthrough:
Frames at constant framerate;
Buffering of frames;
Stalls possible;
Short-term user actions and FoV prediction algorithms assumed.
Li et al. [21]
Key differences:
VR video streaming;
Caching possible
Limited to analytical analysis of offloading paths.
Cheng et al. [4]
Key differences:
Considers VR video streaming;
Fixed offloading ratio: video decoded locally or at the base station;
Hooft et al. [17]
Key differences:
Considers VR video streaming;
Edge not considered
Limited to bitrate adaption scheme (not considering computational offloading)
VR gaming
Guo et al. [16]
Key differences:
Independent Frame-by-frame rendering (excessive rendering delay for one frame does not influence others);
No buffering mechanism;
Caching considered;
Complex decision making based on machine learning.
Mehrabi et al. [28]
Key differences:
Independent Frame-by-frame rendering
Only considers rendering at the cloud and at the edge, not locally;
Ignores local computation delay;
No buffering considered.
Lai et al. [18]
Key differences:
Empirical measurements.
Less powerful hardware considered: mobile phone, Wi-Fi connectivity and offloading to local desktop.

Figure 2.6: Positioning of our work and key differences to state-of-the-art research.

Chapter 3

Analysis of Guo et al.'s Rendering Scheme for VR Games

Guo et al. design a dynamic offloading scheme for VR applications in future MEC-enabled wireless networks based on distributed learning [16]. They focus on the real-time rendering process in wireless VR to maximise the *QoE utility*. The QoE utility stands for the number of users whose end-to-end latency is below the threshold of $\theta_t = \min(1/\eta, \tau)$, where η is the desired framerate and τ the minimal response delay. The main distinguishing aspect of Guo et al.'s work is their sophisticated model that considers the influence of multiple users on each other based on their movements and rendering decisions. Furthermore, Guo et al. devise a complex rendering strategy based on deep reinforcement learning and game theory.

To get insights into the ecosystem of the rendering process in VR games, we analyse this state-of-the-art rendering model in more detail. The leading questions are:

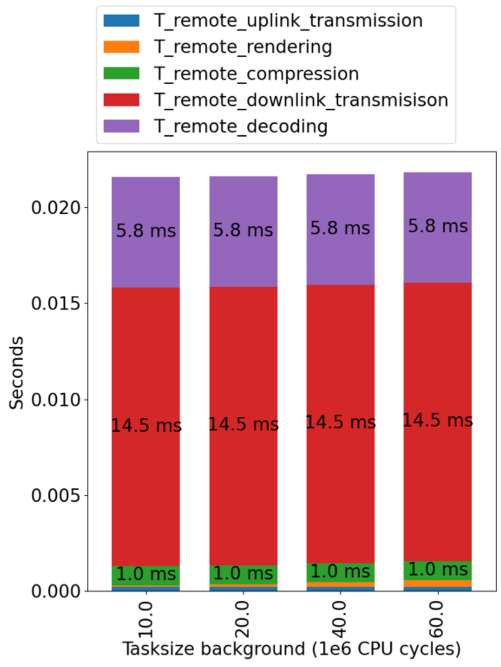
1. What parts of the rendering process take the longest?
2. Can we explain the behaviour and decisions of Guo et al.'s offloading scheme?

The primary motivation for the first question is the following observation. Guo et al.'s performance evaluation shows that with more users, a user's average long-term end-to-end latency increases significantly and even exceeds θ_t. By analysing the intermediate delays in their computing model, we aim to identify these limiting factors to design rendering strategies with better performance. Furthermore, Guo et al. only consider resolutions up to 720p [16], which is significantly lower than expected resolutions for VR [17, 29]. Therefore, we evaluated the performance of Guo et al.'s model under higher resolutions. Secondly, the motivation for the second question stems from the black box that encases the decision-making process of Guo et al.'s algorithm. This obscurity makes it challenging to identify which decisions (offloading paths, resolutions) lead to performance deterioration. The lessons learned during this analysis guide the formulation of the system model in Chapter 4 and help shape the proposed offloading heuristics in Chapter 6.

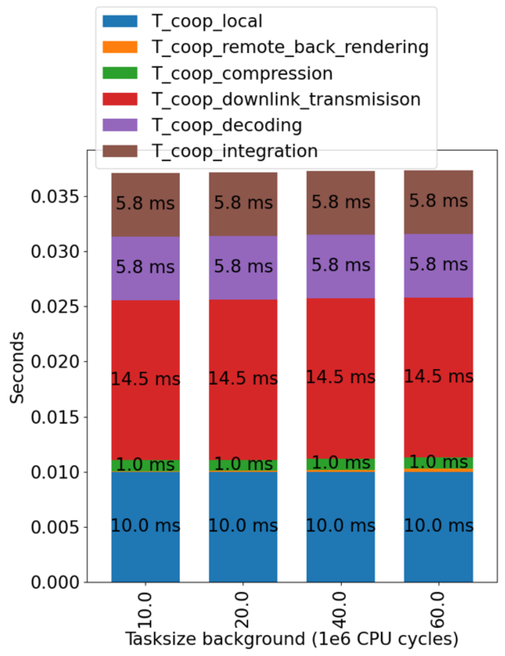
To get insights into the above, we studied Guo et al.'s computing model under different parameters and stored the intermediate delays. Appendix C summarises the formulas and parameter specifications used by Guo et al. [16]. In short, their computing model for the rendering delay when *remotely rendering* T^{remote} consists of delays corresponding to the uplink transmission, the rendering at the MECs, the encoding, the downlink transmission and the decoding. The rendering delay when cooperatively rendering T^{coop} is composed of the integration delay and the maximum between the local foreground and the remote background rendering delay [16]. We took over their simulation values whenever possible. However, We simplified the calculations for the data rate. Instead of calculating the SINR values, we assume a fixed value of 16.5 dB or 20 dB corresponding to good or excellent connectivity in LTE [10]. Moreover, they consider the presence of several users while we limit ourselves to a single user.

Intermediate rendering delays

First of all, the analysis shows that within remote and cooperative offloading, the local processing tasks are limiting factors. Figure 3.1 shows that only the computations at the HMD (decoding, integrating, and the local computations in T^{coop}) significantly affect the total rendering delay next to the downlink transmission. All the remote processing delays are negligible. Note that the *downlink transmission* delay of 14.5ms in Figure 3.1a is the same as in Figure 3.1b because Guo et al. assume that a compressed frame has 2 MB independent of the content [16]. Consequently, $T^{remote} < T^{coop}$ because the integration delay when cooperatively rendering is assumed to be proportional to the uncompressed frame size and easily outweighs the lower rendering delays at the MECs because remote processing is negligible in both cases. Moreover, since decoding (and integration) are assumed to be proportional to the raw frame



(a) Guo et al.'s remote rendering delay T^{remote}.



(b) Guo et al.'s cooperative rendering delay T^{coop}.

Figure 3.1: Intermediate delays when $q_i = 720\text{p}$, $C_f = 10e6$ CPU cycles, $\alpha = 5$ users share resources at the base station and $\gamma = 16.5$ dB.

size, these processing delays become dominant when applying Guo et al.'s formulas to larger resolutions, as shown in Figure 3.2. In this case, the decoding delay alone already exceeds the 16.7ms limit of θ_t.

Table 3.1 shows the local rendering delays of all combinations of processing requirements for the foreground C_f and the background of a frame C_b assumed by Guo et al. [16]. Under all these combinations, only 2 out of 16 fall under the θ_t threshold showing that local rendering is hardly advisable. Comparing the values of T^{local} when $C_f = 10e6$ CPU cycles to the decoding delays in Figure 3.2, we can observe that the decoding delay of 51.8ms even exceeds the local rendering delay when $C_B \neq 60e6$ CPU cycles, implying that decompressing a frame takes longer than rendering the frame.

This observation seems counter-intuitive, which prompted us to deviate from Guo et al.'s data and

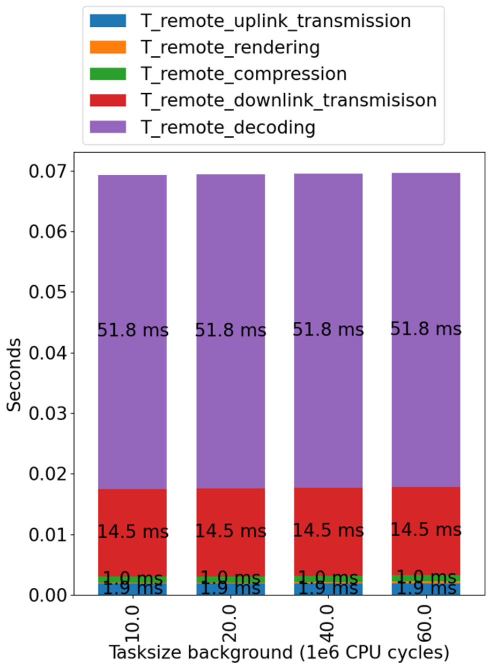


Figure 3.2: Intermediate delays for Guo et al.'s remote rendering delay T^{remote} when $q = 2160$p, $C_f = 10e6, \alpha = 5$ users share resources at the base station and $\gamma = 16.5$ dB.

computing model. In fact, we believe the origin of the observation stems from the following modelling assumptions of Guo et al. [16]:

- The data size D and computational requirement C to render the frame are not explicitly linked.
- The decoding delay of a frame at the HMD is proportional to the data size of the corresponding raw frame.

Both assumptions together result in the fact that the local rendering delay $(C_f + C_b)/z_l$ is smaller than the decoding delay $D/(b_{zl} \cdot z_l)$ for the values specified beforehand. However, it is impossible to make the above conclusions with certainty because Guo et al. do not describe clearly how the data sizes and computational requirements vary between frames.

Possible decision-making process of Guo et al.'s scheme

The second main goal of the analysis is to explain the decision-making process of Guo et al.'s offloading scheme. For the remaining calculations, we emphasise that Guo et al. do not specify how they choose the quality q or the computational requirements C_f and C_b of a frame, so we assume that all combinations are equally likely.

Let θ_t be the threshold of the QoE-utility value: the QoE utility equals 1 if the rendering delay is below θ_t and 0 if it is above and let the channel quality be fixed. Then, by transforming the equations of Guo et al.'s computing model [16], we can calculate the number of people that can simultaneously render remotely such that everyone's rendering delay is below θ_t. The formulas are given in Appendix C.2 together with the summarising figure showing the maximum number of users for the different parameter combinations. On average, under *excellent* SINR (γ = 20 dB), over Guo et al.'s simulated computational

Table 3.1: Local rendering delay T^{local} under Guo et al.'s model specifications for local rendering when $z_l = 1e9$ CPU cycles/s and $\theta_t = \min(1/60, 0.025) = 16.7\text{ms}$.

C_b (1e6 CPU cycles)	C_f (1e6 CPU cycles)	T^{local} (ms)	$T^{local} \leq \theta_t$?
10	2.5	12.5	yes
10	5	15	yes
10	10	20	no
10	20	30	no
20	2.5	22.5	no
20	5	25	no
20	10	30	no
20	20	40	no
40	2.5	42.5	no
40	5	45	no
40	10	50	no
40	20	60	no
60	2.5	62.5	no
60	5	65	no
60	10	70	no
60	20	80	no

requirements and resolutions,

$$(\underbrace{5.8 + 10 \cdot 5.7 + 5 \cdot 5.6}_{\text{360p}} + \underbrace{9 \cdot 5.3 + 7 \cdot 5.2}_{\text{480p}} + \underbrace{4 \cdot 4.1 + 12 \cdot 4.0}_{\text{720p}})/(3 \cdot 16) = 4.99 \approx 5$$

users can simultaneously render remotely for each of them to have a QoE-utility of 1. 16 stands for the number of possible combinations of C_f and C_b since both can take on four different values.

Now, consider a distributed offloading strategy that ensures that maximally five users render remotely per base station and all other users render locally. Then, the QoE-utility will be 1 for the five users offloading remotely. For local offloading, Table 3.1 shows that the delay $(C_b + C_f)/z_l$ is below θ_t in 2 out of 16 times, so with a probability of 0.125. Thus, whenever there are 100 users and 10 base stations (as assumed in Guo et al.'s performance evaluation [16]), the expected average QoE-utility is:

$$\text{Expected QoE-Utility} = \underbrace{5 \times 10 \times 1}_{\text{Users choosing remote rendering}} + \underbrace{(100 - 50) \cdot 0.125}_{\text{Users choosing local rendering}} = 56.2.$$

Comparing this to the long-term average QoE-utility of Guo et al. proposed algorithm, we can observe that its QoE-utility of 65 is higher, so the above cannot fully explain the algorithm's decision-making. Of course, it is challenging to estimate the exact QoE-utility because we cannot simulate the strategy under the exact setting as Guo et al. used. With the above, we must conclude that we cannot fully explain which decisions are made by Guo et al.'s algorithm nor what rendering decisions lead to a positive QoE-utility. This observation encourages us to design offloading heuristics whose behaviour is fully explainable to determine why dynamic strategies might be superior to static strategies.

Chapter 4

System Model and Problem Formulation

In an edge-enabled network, a single user is playing a virtual reality game. As shown in Figure 4.1, a base station provides connectivity and multi-access edge computing (MEC) to a single user playing an immersive VR game on a HMD. The HMD connects to the base station via a wireless connection that offers 5G and beyond data rates and latencies.

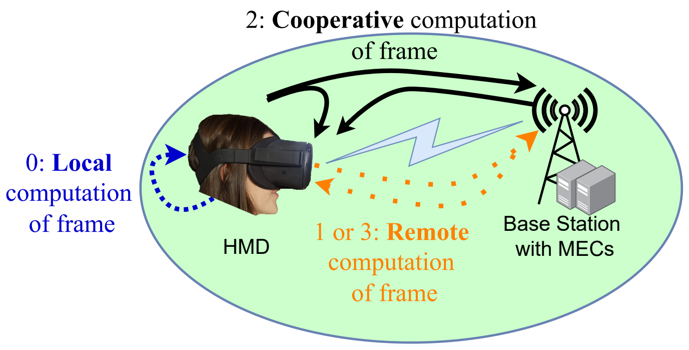

2: Cooperative computation of frame
0: Local computation of frame
HMD
1 or 3: Remote computation of frame
Base Station with MECs

Figure 4.1: System Overview showing the three rendering paths.

For a good user experience, the HMD needs to deliver VR frames to the user at a satisfying framerate, a high quality and without interruptions. The rendering strategy of the HMD makes the **rendering decision** on

1. which quality resolution to choose? (Bitrate decision)

2. which offloading path to choose? (Offloading decision)

Note that for every offloading path, the rendering delay (also called offloading delay) depends on the chosen quality resolution, so the bitrate and the offloading decision must be jointly considered. The remainder of this chapter describes the different components needed to translate this setting into a system model:

4.1: Building blocks: how can the individual components and their interactions be mathematically described?

4.2: Modelling the communication between edge and HMD: what data rate can be reached?

4.3: Modelling the structure and size of a frame: how large are VR frames of a given resolution?

4.4: Modelling the rendering process: what is the rendering delay per path?

4.5: Modelling the display process: how quickly does the HMD display frames?

4.6: Modelling the quality-of-experience of the user: by which performance metrics can we measure a "good experience" ?

Finally, we present the underlying optimisation problem in Section 4.7.

4.1 Building blocks

Figure 4.1 depicts the overall system. A *single* base station (BS) provides MEC services to a user engaged in a VR game using a HMD and controllers. While multiple users are present, the point of view of the study only follows a single user and abstracts other users to a number α that affects the user's share of bandwidth and processing power. To clarify, the available bandwidth to the user is B/α of the total available bandwidth B. Similarly, let Z_C be the total processing power at the MECs, then the user can receive a fair share, namely $z_c = \frac{Z_C}{\alpha}$ [16].

We *assume* that a user is playing a VR game with the following properties. We use the terminology *video* and *game* as follows. The user watches a *video* on their HMD. This video is influenced by the user's play actions and head movement (the latter determines the FoV). The overall process is called *game*.

1. **VR game**. We assume the user is playing a VR game in which the background content is highly predictable (video-like) while the foreground contains all user interactions.

2. **Data flow**. The entire game can be played locally (at the HMD device) with optional computational offloading to MECs requiring the following prerequisite. Similarly to the assumption of [16], all model information needed to render the game content is present at the MEC server and the HMD. To render a frame, some uplink communication (for example, user position and user actions) is necessary, but it is assumed small.

3. **Duration**. The user can, in principle, play the game at will. For our model, we *assume* that the player is engaged in the game for a fixed time T.

4. **Playback**. If possible, the game is played at a *fixed* framerate of 60 FPS (frames per second). If there is no frame ready to be displayed when needed, the video pauses until the next frame is ready. This occurrence is called *stalling*.

5. **Start-up delay**. At the beginning of the game, a fixed number of frames are pre-rendered before displaying frames to the user.

6. **Video $\ni$ frames $\ni$ tiles $\ni$ FoV**. Time-wise, the video game is divided into a sequence of *frames*, each depicting an image of the sequence. Furthermore, every *frame* is spatially split into *tiles*. A user does not see the entire frame but only the tiles lying in their Field of View (further explanations on FoV below). Figure 2.3 illustrates this. We *assume* that all tiles and frames can be rendered separately and at different resolutions.

To focus on the effect of the rendering decision on the delay, we abstract user input by assuming the existence of predefined, accurate user prediction models. The user influences the system in two ways: by determining what they want to see and by their interactions when playing the game:

- **Field of View (FoV).** The 360° image of a VR application is larger than the human visual range [14]. Consequently, a user only sees parts of the overall image depending on their head position. The set of tiles that lies in the user's viewpoint is called *Field of View (FoV)*. Figure 2.3 shows an example of the spatial tiling of a VR video of which 4 out of 24 tiles lie in the FoV.

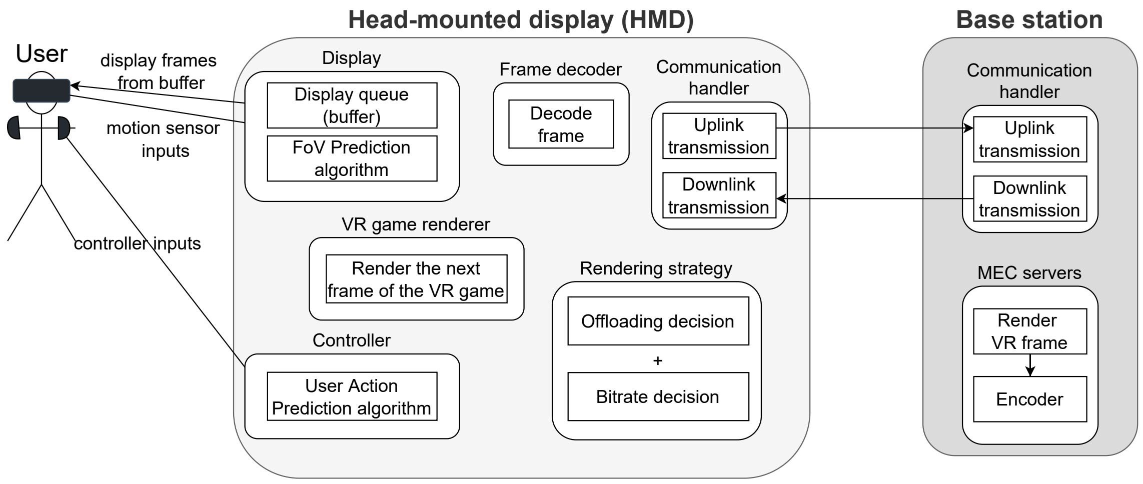


Figure 4.2: Overview of all components of the HMD and MEC servers.

- **User Action.** Since we study a VR game, user interactions are crucial. Depending on the user input, the content of the frames changes. We *assume* that the short-term user actions only influence the foreground information and not the background information. For example, whether a user lets an apple fall onto the ground or eats an apple does not influence the rendering of the grass, sky and mountains in the background.

We *assume* the presence of a *perfect* FoV prediction algorithm, as well as a *perfect* user action prediction algorithm. Then, we can model the application with streaming aspects: the HMD has all information necessary to render the frames at the moment of the rendering decision, producing a stream of images similar to the contents downloaded when streaming a 360° video. In this way, we can study the clean potential of the rendering decision uninfluenced by other algorithms to predict the FoV or user actions.

The HMD is the connection between all other entities. Figure 4.2 presents an overview of its various components and tasks. First, it registers user inputs using sensors and controllers. These are used to calculate (predict) the FoV and the contents of the frame (digital consequences of user actions). Furthermore, it displays ready-frames at the desired framerate to the display of the HMD and maintains a constant connection to the base station. Lastly, it is responsible for rendering the frames and making the *rendering decision*.

The *rendering decision* consists of the bitrate decision and the offloading decision. The former determines the resolution of the tiles lying in the user's FoV. The latter decides *how* to compute the frame, as the HMD can decide to offload the rendering task to the MECs. If it decides to render *locally*, the HMD itself renders the frame based on the (predicted) user actions and FoV and stores the result in the buffer until it is ready to be displayed. When rendering *remotely*, it transmits all necessary information to the MECs and downloads the rendered frame after the completion of the rendering task by the MECs. If the MECs encode the image, the HMD will first decode the frame before adding the ready frame into the buffer. Lastly, when rendering *cooperatively*, the HMD will locally render the frame's foreground according to user actions and FoV, download the rendered background and then combine both into the final frame.

We *assume* that the rendering process stops after a frame has been dropped due to a full buffer and restarts once displaying a new frame frees buffer space. During the above, we *assume* that all non-rendering delays, such as making the decision or displaying frames, are negligible (for example, they could be done in parallel) and denote the available processing power of the HMD by z_l. We *consider* the unit of the buffer to be *number of frames*, implying that the total size of the buffer in bits can be different even if there is an equal number of frames in the buffer. Similarly, the maximum buffer capacity is expressed as the maximal number of frames it can hold.

4.2 Communication model

Transmitting information from the HMD to the base station and back happens over a wireless channel using 5G and beyond technology. According to Shannon's capacity formula, the data rate R depends on the available bandwidth B, how the bandwidth is shared among α users, and the Signal-to-Interference-and-Noise ratio, $\gamma = \frac{S}{N}$ [16]. The latter ratio indicates the strength of the signal S relative to ambient noise and interference N. We emphasise that the data rate depends not on the absolute signal strength but the relative strength against the noise and interference. Then, assuming equal bandwidth allocation between all α users, the data rate R is calculated by

$$R = \frac{B}{\alpha} \log_2(1 + \gamma_l). \tag{4.1}$$

Similarly to Mehrabi et al. [28], we consider γ an input parameter. This generalisation simplifies the communication model and allows for future incorporations of more complex models to calculate SINR values. Note that we express the SINR γ in decibel (dB), so the following conversion is done before applying the value to Equation (4.1):

$$\gamma_l = 10^{\frac{\gamma}{10}},$$

We differentiate between uplink and downlink by using the superscript u for uplink and d for downlink: $R^u, R^d, B^u, B^d, \gamma^u$ and γ^d.

Consequently, the transmission delay to communicate data of size D is calculated by D/R. We assume a negligible propagation delay due to the proximity between base stations and users in 5G and beyond networks.

4.3 Data model: quality resolutions, data sizes and computational requirements

A frame's raw *data size* D is defined as the memory needed to store the frame and measured in bits. A frame's *computational requirement* C stands for the processing needed to render the frame and is measured in CPU cycles. Both C and D depend on the desired quality resolution q of the frame (e.g 360p, 480p, etc).

Every frame is spatially split into $l \times w$ equally sized tiles, which all can be requested at different resolutions. The total data size D and computational requirement C of a frame are calculated by summing the sizes D_t and requirements C_t of all tiles in the frame, respectively. Furthermore, a tile consists of a background and a foreground which can be computed separately and whose sum equals the size or requirement of a tile:

$$D_t = D_b + D_f, \qquad C_t = C_b + C_f, \tag{4.2}$$

where the subscript b stands for the background and f for the foreground of a tile.

First, we describe the calculations of the raw *data sizes* from the requested resolution q and the number of tiles $l \times w$. Given a picture at qp, the short side of the picture consists of q pixels. Assuming an aspect ratio of 16:9 [5], the longer side has $16/9 \cdot q$ pixels. This work assumes that every pixel requires 10 bits for deeper colour storage implying that a full frame at qp has a data size of $16/9q^2 \cdot 10$ bits.

Furthermore, since we assume a spatial tiling of $l \times w$ of every frame, a single tile has a size of $\frac{16/9q^2 \cdot 10}{l \cdot w}$ bits. We *assume* that the background is richer and more detailed. Formally, a tile consists for β out of background and $1 - \beta$ out of foreground with $\beta \in [0, 1]$. Next, when rendering (parts of) the frame at the MECs, the HMD does not need to communicate a full frame to the base station, but only the information necessary to render it, such as the (predicted) user actions, the requested quality resolution and the FoV. We consider the data size to be sent proportional to the full frame. Similarly to Guo et al. [16], we assume that the uplink data size to send is 1/1000 of the full frame size. Then, the data sizes for a tile are calculated as follows:

$$
\begin{aligned}
D_t(q) &= D_f(q) + D_b(q) = \frac{16/9q^2 \cdot 10}{lw}, && \text{(Data size of tile in bits)}\\
D_f(q) &= (1-\beta) \cdot D_t(q) = \frac{(1-\beta) \cdot 16/9q^2 \cdot 10}{lw}, && \text{(Data size of foreground in bits)}\\
D_b(q) &= \beta \cdot D_t(q) = \frac{\beta \cdot 16/9q^2 \cdot 10}{lw}, && \text{(Data size of background in bits)}\\
D_u(q) &= D_f(q)/1000 = \frac{(1-\beta)16/9q^2 \cdot 10}{1000lw}, && \text{(Uplink data size in bits)}
\end{aligned}
$$

with $l \times w$ the number of tiles in a frame (length $\times$ width), q the requested resolution and β the proportion of the frame that belongs to the background.

The *computational requirements* of a tile are calculated as follows. Guo et al. assume that both the HMD and MECs can process 0.4 bits per CPU cycle [16]. Hence 1 CPU cycle is equivalent to 0.4 bits and we translate this relationship to the data sizes and computational requirements to get D (bits) $= 0.4$ (bits/CPU cycle) $\cdot$ C (CPU cycles). Applying this relation to the above data sizes gives:

$$
\begin{aligned}
C_t(q) &= C_f(q) + C_b(q) = \frac{D_t(q)}{0.4} = \frac{16/9q^2 \cdot 8}{0.4lw}, && \text{(Computational requirement of tile in CPU cycles)}\\
C_f(q) &= \frac{D_f(q)}{0.4} = \frac{(1-\beta)16/9q^2 \cdot 8}{0.4lw}, && \text{(Computational requirement of foreground in CPU cycles)}\\
C_b(q) &= \frac{D_b(q)}{0.4} = \frac{\beta 16/9q^2 \cdot 8}{0.4lw}, && \text{(Computational requirement of background in CPU cycles)}
\end{aligned}
$$

with the number of tiles in a frame $l \times w$ (length $\times$ width), the requested resolution q and the proportion of the frame that belongs to the background β.

Finally, the raw data size D and computational requirement C of the full frame is:

$$
\begin{aligned}
D &= \sum_{\text{tile } t} D_t,\\
C &= \sum_{\text{tile } t} C_t.
\end{aligned}
$$

When referring to a specific frame i, all symbols have an additional superscript i, such as C^i and D^i.

To give an idea about the resulting uncompressed bit-streams at a framerate of 60 FPS, consult Figure 4.3. VR applications generally aim for high quality to provide a realistic immersion. This means

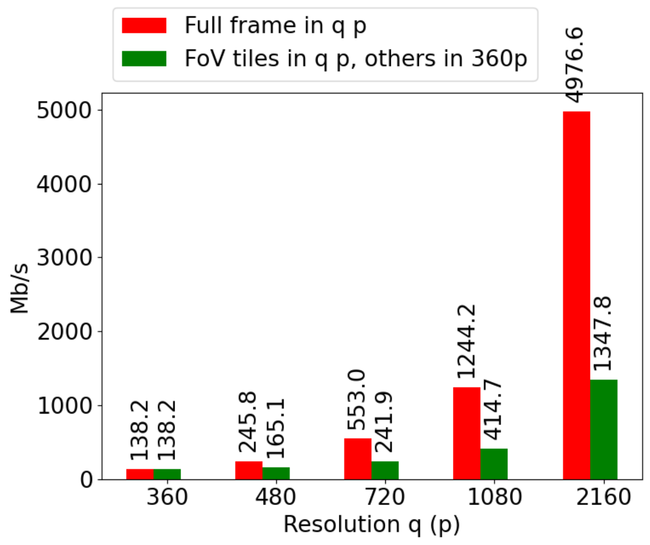


Figure 4.3: Bitrates for different quality resolutions q.

that resolutions such as ultra-high-definition (UHD) at 2160p are not uncommon. To support UHD, bitrate adaption schemes render tiles in the FoV at higher quality than the remaining tiles [17]. Figure 4.3 shows that this technique significantly reduces the application's bit-stream. To further reduce the bit stream over the network, algorithms use compression algorithms to decrease the file size of individual images. In case compression is used, the data size and reduces by a compression ratio factor k:

$$D_k = \frac{1}{k} \sum_{\text{tile } t} D_t,$$

$$C_k = \frac{1}{k} \sum_{\text{tile } t} C_t.$$

Video compression is also called encoding, and video decompression is called decoding.

4.4 Computing model

The rendering of a frame i depends on the rendering decision $\phi = (o, q_i)$. More concretely, it depends on the requested resolution q_i to determine a frame's size and computational requirements and the chosen offloading path o. Rendering a frame can happen locally at the HMD (local), without encoding and decoding at the MECs (remotely), at both (cooperative), or with encoding at the MECs (remote encoding). Let $\mathcal{O} = \{0, 1, 2, 3\}$ denote the *mode selection indicator set*, i.e the set of all possible offloading decisions, with

- 0: local rendering;
- 1: remote rendering;
- 2: cooperative rendering;
- 3: remote encoding, remote rendering with encoding and decoding.

Rendering of the gaming application happens frame-by-frame. As explained in Section 4.1, every frame consists of a set of tiles of which a subset is contained in the user's FoV. We assume that the game is displayed at a fixed 'screen' resolution; hence, the number of tiles in every frame is fixed to $l \times w$. Furthermore, we assume that the number of tiles within the FoV is fixed (i.e. the minimum number of tiles that can cover the FoV at all times) and set to n. Additionally, we assume that every tile can be rendered at a different resolution.

Let q_0 be the lowest resolution and q_i the requested FoV resolution according to strategy ϕ. Then, the n tiles in the FoV are rendered at quality q_i, while the other $l \times w - n$ tiles are rendered at quality q_0. As specified in Section 4.3 the resulting data size D and computational requirement C of a frame is:

$$D = n \cdot D_t(q_i) + (l \times w - n) \cdot D_t(q_0), \tag{4.3}$$

$$C = n \cdot C_t(q_i) + (l \times w - n) \cdot C_t(q_0). \tag{4.4}$$

Depending on the selected mode (local, remote, cooperative or remote encoding), the task computation and corresponding delay have different components listed below.

4.4.1 Local rendering

In *local rendering*, the entire frame is rendered locally at the HMD. Hence the computation delay T_{local} only depends on the frame's computational requirement C and the HMD's processing capability z_l:

$$T_{local} = \underbrace{\frac{C}{z_l}}_{\text{render fore- and background}} . \tag{4.5}$$

4.4.2 Remote rendering

In *remote rendering*, both fore- and background of a frame are rendered remotely at the base station. For the base station to render the dynamic foreground, it needs to receive some information from the VR device; D_u denotes the volume of this data. The latency for remote rendering consists of the delay for the uplink transmission, the rendering of the frame, and the downlink transmission delay:

$$T_{remote} = \underbrace{\frac{D_u}{R^u}}_{\text{Uplink transmission}} + \underbrace{\frac{C}{z_c}}_{\text{render fore- and background}} + \underbrace{\frac{D}{R^d}}_{\text{Downlink Transmission}} , \tag{4.6}$$

where R^u and R^d are the up- and downlink transmission rates of the user, D and C the data volume and size of the frame, D_u the data volume needed for the uplink communication and z_c the processing power at the MECs allocated to the user.

Compared to Guo et al.'s model [16], we *assume* that the frames are not cached and not encoded between the base station and HMD. The former is a model simplification, and the latter results from our analysis of Guo et al.'s model: in Chapter 3, we show that decoding the file locally is a significant delay in their model and comparable to rendering the frame in itself. Cheng et al. engaged in a similar idea in their streaming model where the offloading decision determines what percent of the frame to already decode at the base station [4], instead of decoding the entire frame at the HMD.

4.4.3 Cooperative rendering

In *cooperative rendering*, the foreground of a frame is rendered locally at the VR end device, while the background is rendered at the base station. Since back- and foreground are computed separately, the

HMD needs to eventually integrate both parts into a final frame. According to [18], this action comes down to a copying instruction making the delay negligible. We regard this *integration delay* $t_{integrate}$ as a constant and set it to 1ms, the empirical measurement of Lai et al. [18].

$$T_{coop} = \max\left(\underbrace{\frac{C_f}{z_l}}_{\text{local foreground rendering}} , \underbrace{\underbrace{\frac{D_u}{R^u}}_{\text{Uplink Transmission}} + \underbrace{\frac{C_b}{z_c}}_{\text{Remote background rendering}} + \underbrace{\frac{D_b}{R^d}}_{\text{Downlink Transmission}}}_{\text{Remote background rendering}} \right) + \underbrace{t_{integrate}}_{\text{integrate fore- and background}} , \quad (4.7)$$

where z_c is the processing power at the base station that is allocated to the user, C_f and C_b are the computational requirements of the fore- and background of the frame, D_b is the associated data size of the frame's background, R^u and R^d the up- and download transmission rates and $t_{integrate}$ the integration delay constant. We do not consider encoding and decoding with the same reasoning as in Section 4.4.2.

4.4.4 Remote rendering with encoding

Remote rendering with encoding and decoding is comparable to current streaming practices. Content is readied at a remote content provider (often the cloud) and encoded before communicating it over the network. Conceptually, encoding uses an image or video's intrinsic structures and patterns to reduce the file size, which is also known as *compressing* a file. After downloading the encoded frame, the HMD must decode the video to display the uncompressed version. This practice trades transmission delay against computation delay because it reduces the amount of data in transit at the expense of additional computations at the endpoints.

We *assume* that encoding a frame requires processing of the entire *uncompressed* frame and decoding requires processing of the *compressed* frame. The size relation between compressed and uncompressed files is denoted by the compression coefficient k. We acknowledge that modelling precise encoding and decoding values is tricky and highly dependent on the compression algorithm, the characteristics of the content and the hardware used. Lai et al. empirically measure the encoding and decoding delays and the compression ratio when running three high-end VR apps on a mobile phone connected to a desktop server [18]. For example, using H.264, the server needs 10 ms each to render and encode the video of the "Corridor" app, while the phone needs 16ms to decode it [18]. As the desktop's processing power is at least twice as large as the phone's, their empirical numbers support the claim that encoding is comparable to processing the entire frame size and decoding only needs to process a smaller file. All other intermediate delays are the same as in Section 4.4.2:

$$T_{remoteenc} = \underbrace{\frac{D_u}{R^u}}_{\text{Uplink transmission}} + \underbrace{\frac{C}{z_c}}_{\text{render fore- and background}} + \underbrace{\frac{k_{enc} \cdot D}{z_c \cdot b_{z_c}}}_{\text{Compression latency}} + \underbrace{\frac{D_k}{R^d}}_{\text{Downlink Transmission}} + \underbrace{\frac{D_k}{z_l \cdot b_{z_l}}}_{\text{Decoding compressed frame}} , \quad (4.8)$$

where R^u and R^d are the up- and downlink transmission rates of the user, D and C the data volume and size of the frame, D_u the data volume needed for the uplink communication about the foreground rendering, $D_k = \frac{D}{k}$ the file size of the compressed frame depending on the compression ratio k, z_c the processing power at the MECs allocated to the user and b_{zc} (b_{zl}) the number of bits the MECs (the HMD) can process per CPU cycle.

4.5 Playback model

The HMD plays out content at a fixed framerate of η frames per second if the buffer is non-empty. If the buffer is empty, the next frame is displayed as soon as it is available. We *assume* displaying a frame to the user does not take away processing resources from the rendering process and can happen in parallel. We justify this assumption by assuming that the HMD has a dedicated part of its processing power reserved for core functionalities, and the remaining processing power z_l is available for the rendering process.

4.6 Quality of Experience model: QoE-score

Measuring a user's experience when playing a VR game goes beyond measuring the quality of service (QoS) agreement on a networking level. Even if QoS requirements are met, a user can still have a bad quality of experience (QoE). For example, the link latency could be below the service-level threshold, ensuring that the link communication was sufficiently quick. However, if the HMD needs too long to process additional tasks (such as decoding), the end-to-end delay between requesting a frame and displaying the frame could still be too long. Conversely, the same holds. Even if QoS requirements are not guaranteed, the HMD could ensure a constant framerate by employing buffers and making good offloading choices on when to use the network and when to compute locally. Hence, we would like to compare our proposed offloading strategies on a single *QoE-score* encompassing the main measurable QoE metrics involved in VR streaming. As proposed by [7], we consider *quality*, *stalling*, *quality switches* and the *start-up delay*. Differently from their model, we normalise the individual components to bring the values into the same order of magnitude. The goal is to maximise the quality while reducing stalling, quality switches and start-up delay.

First, the *QoE-quality* is the division between the effective bitrate θ and the maximally achievable bitrate Θ. The maximal bitrate Θ corresponds to continually rendering the FoV of frames at the highest resolution (2160p) while supporting the desired framerate of 60 FPS, resulting in a raw bitrate of 1348 Mbits/s as shown in Figure 4.3. Moreover, let N_T be the total number of frames played to the user. Then:

$$D_{played} = \sum_{i=1}^{N_T} D^i, \qquad \textit{(Sum of displayed bits)} \tag{4.9}$$

$$\theta = \frac{D_{played}}{T - T_0}, \qquad \textit{(Effective bitrate)} \tag{4.10}$$

$$\text{QoE-quality} = \frac{\theta}{\Theta}. \tag{4.11}$$

Secondly, the *QoE-start-up* is the time needed between starting the game and b frames being buffered divided by the maximally accepted start-up time. We set this maximum to 1s. According to [35], start-up delays are a significant factor in why users abandon a video stream. However, we argue that users have a higher tolerance for start-up delays when playing a VR game since they already made a more significant time dedication by putting on the HMD and installing a specific game. Hence:

$$\text{QoE-start-up} = \frac{T_0}{1} = T_0. \qquad \textit{(Time until b frames are buffered)} \tag{4.12}$$

Next, *QoE-stall* is the percentage of the total playtime during which the system was in a stall. Mathematically, this translates to:

$$T_S = T - T_0 - N_T \cdot \frac{1}{\eta}, \qquad \textit{(Total stall duration)} \tag{4.13}$$

$$\text{QoE-stalls} = \frac{T_S}{T}; \tag{4.14}$$

with T the total play time, T_0 the start-up delay, N_T the total number of frames played and η the desired framerate. Last but not least, the *QoE-switches* metric is the sum of all bits involved in a quality switch divided by the sum of all bits played:

$$D_{played} = \sum_{i=1}^{N_T} D^i, \qquad \textit{(All bits played)} \tag{4.15}$$

$$D_{switches} = \sum_{i=2}^{N_T} |D^i - D^{i-1}|, \qquad \textit{(Bits involved in quality switch)} \tag{4.16}$$

$$\text{QoE-switches} = \frac{D_{switches}}{D_{played}}, \tag{4.17}$$

with N_T the total number of frames played to the user and D^i the data size of frame i. By basing the QoE-switches metric on the difference between frame sizes, we can measure the 'severeness' of a quality switch. For example, switching between 2160p and 720p is more severe than a switch from 360p to 480p because the number of pixels on the screen changes more significantly in the former case. Finally, the *QoE-score* is calculated by:

$$\text{QoE-score} = \kappa_1 \cdot \text{QoE-quality} - \kappa_2 \cdot \text{QoE-stalls} - \kappa_3 \cdot \text{Quality-switches} - \kappa_4 \cdot \text{QoE-start-up}, \tag{4.18}$$

$\kappa_1, \kappa_2, \kappa_3$ and κ_4 are coefficients that can tweak the influence of the individual components. For the remainder of the document, we assume that all components are equally important and set the coefficients to 1.

Moreover, while we desire a single value to compare the offloading strategies in a single glance, we also want to evaluate them on other performance metrics. The complete overview of performance metrics considered in the performance evaluation of the proposed offloading strategies is given in Section 8.1.

4.7 Problem formulation

The main goal of our research is to find a rendering strategy ϕ that optimises the QoE experience of a user playing a VR game. The rendering decision ϕ decides for every frame $i \in \{1, 2, \ldots, N_T\}$, at what quality resolution $\mathcal{Q}(i) = q_i$ the frame should be rendered and which offloading path $\mathcal{O}(i) = o_i$ to take. Let $\mathcal{O}$ be the sequence of chosen offloading modes and $\mathcal{Q}$ the sequence of chosen resolutions. Then, for every frame i, the offloading strategy ϕ decides the offloading mode o_i and the resolution q_i. In symbols, $\phi(i) = (\mathcal{O}(i), \mathcal{Q}(i)), \forall i$. Then, the optimisation problem can be formulated as follows:

$$\max_{\phi} \left(\underbrace{\frac{\sum_{i=1}^{N_T} D^i}{\Theta}}_{\text{QoE-quality}} - \underbrace{\frac{T - T_0 - N_T \cdot \eta}{T}}_{\text{QoE-stalls}} - \underbrace{\frac{\sum_{i=2}^{N_T} |D^i - D^{(i-1)}|}{\sum_{i=1}^{N_T} D^i}}_{\text{QoE-switches}} - \underbrace{T_0}_{\text{QoE-start-up}} \right) \tag{4.19}$$

with

$$\phi(i) = (O(i), Q(i)), \forall i \in [0, 1, 2, \ldots, N_T], \qquad \textit{(Rendering strategy)} \tag{4.20}$$

$$\mathcal{O}(i) \in \{0, 1, 2, 3\}, \qquad \textit{(Offloading mode for frame i)} \tag{4.21}$$

$$\mathcal{Q}(i) \in \{360p, 480p, 720p, 1080p, 2060p\}, \qquad \textit{(Quality resolution for frame i)} \tag{4.22}$$

$$T_0 \leq T, \qquad \textit{(Start-up delay)} \tag{4.23}$$

$$N_T \leq \eta \cdot T, \qquad \textit{(Number of frames played in time } T\textit{)} \tag{4.24}$$

with N_T the number of frames played, T the play duration, η the desired framerate and Θ the maximal bitrate possible and κ_1, κ_2, κ_3 and κ_4 are coefficients of the optimisation problem to tweak the influence of the individual components.

This optimisation problem depends on the randomness of the channel quality at different moments in time and on other system parameters that we consider fixed, such as the bandwidth or the available processing power of the MECs or the HMD. The optimisation problem is difficult to solve analytically because the rendering of earlier frames influences the stalling probability of later frames. Moreover, the rendering and display loops happen simultaneously, which is problematic to accurately model in a mathematical model.

Chapter 5

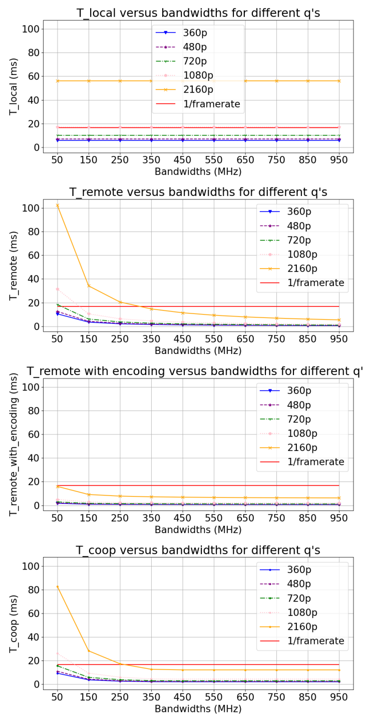
Analysis of our System Model

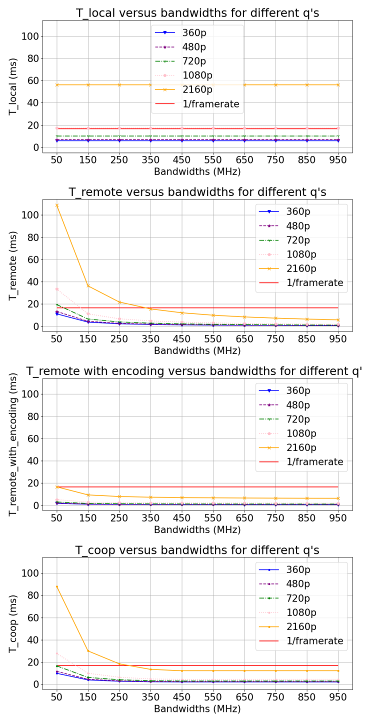
This chapter analyses the offloading paths under the system model proposed in Chapter 4. More concretely, the offloading delays in Equations (4.5), (4.6), (4.7) and (4.8) are studied under different system parameter values. To design offloading heuristics and choose adequate system parameters, we first seek better insights into the system's behaviour. This chapter aims to answer the following questions:

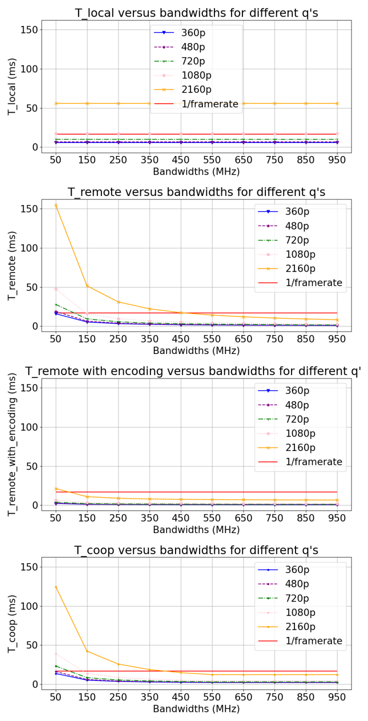
1. Under what bandwidth and SINR conditions is offloading feasible? In other words, under what circumstances are T_{local}, T_{remote}, T_{coop} and $T_{remoteenc}$ below the threshold $1/\eta = 1/60$ s $= 16.7$ ms needed to support a framerate of $\eta = 60$FPS?
2. What is the maximum number of people that can be served in the system? We define the maximum number of people, denoted by M_{α}, as the number of people that can simultaneously offload to the MECs using the same offloading path.
3. What are typical offloading delays? *Typically* meaning under reasonable bandwidth and SINR conditions that we could expect when playing a VR game.

First, under three different SINR values, the offloading delays are computed over up- and downlink bandwidth ranging from 50MHz to 1000 MHz. This range allows us to consider next-generation technologies such as 5G and beyond. The SINR values represent an *excellent* connection, when $\gamma = 20$ dB, a *good* connection when $\gamma = 16.5$ dB or a *mid-cell* connection when $\gamma = 6.5$ dB [10]. The resulting offloading delays are depicted in Figure 5.1. The local rendering delay naturally does not change with the bandwidth as it only consists of the delay to render the frame at the HMD. The main conclusion from the first row of these figures is that T_{local} only falls below the desired threshold of 16.7 ms when the resolution $q \leq 720$p. Hence, when constantly computing locally, the highest quality to choose without risking stalls is 720p. A general observation regarding all other offloading delays that make use of the network is that the delays increase between Figures 5.1a, 5.1b and 5.1c as the SINR value worsens but still fall under the 1/60s threshold eventually for every resolution. Note that we assume that the user can fully use the entire bandwidth. In practice, even with 5G mmWaves providing Gbit-data rates, an effective bandwidth of 450MHz as needed for T_{remote} under mid-cell circumstances is likely to be unrealistic for the foreseeable future since a user needs to share the bandwidth with other users. Converting this to data rates, to remotely render a frame at 2160p, the user requires a bandwidth of 300MHz, which equals a data rate of $300e6 \cdot \log_2(1 + 100) \approx$ 2Gbps, which is still beyond the promised 1 Gbps of 5G. Hence, we would require offloading heuristics that can consider the buffered frames to remotely or cooperatively render frames at 2160p.

The remote offloading with encoding delay $T_{remoteenc}$ is by far the lowest delay over all SINR and bandwidth values. Except for a bandwidth of 50MHz, $T_{remoteenc}$ is also below the 1/60s threshold,







(a) Excellent SINR of 20dB. (b) Good SINR of 16.5dB. (c) Mid-cell SINR of 6.5dB.

Figure 5.1: Offloading delays over different bandwidths for fixed SINR values.

making remotely offloading with encoding very promising. However, we must be cautious because, in Chapter 3, we have shown that the decoding delay is a limiting factor. Different from the model by Guo et al. in [16], we assume the decoding delay is also proportional to the compression ratio k. To be precise, in our model, we assume that the HMD only needs to process data of size D/k, the size of the compressed frame. The last observation is that at 150 MHz, T_{remote} and T_{coop} when rendering at 1080p are below the threshold of 1/60 s.

The above numbers are all under the assumption that the user can hoard the entire bandwidth and processing power of the MECs. In reality, these resources are shared between users. For the remainder of the section, we assume a bandwidth of 150MHz, which by Shannon's formula corresponds to a bitrate of almost 1 Gbps, the promised data rates available to users in 5G and beyond technologies. Figure 5.2 shows how the offloading delays develop when additional users are connected to the base station, all getting an equal share of the bandwidth B and the processing power Z_C. Remarkably, even rendering at the lowest resolution 360p does not ensure that T_{remote} or T_{coop} stay below the 1/60 s framerate if the number of users exceeds 5. Analytically, we determined the maximal number of people M_α that can be accommodated at the base station for the different offloading paths:

$$\begin{aligned} T_{remote} \leq \frac{1}{\eta} &\iff \frac{D_u}{\frac{B^u}{M_{\alpha,remote}}\log_2(1+\gamma)} + \frac{C}{\frac{Z_C}{M_{\alpha,remote}}} + \frac{D}{\frac{B^d}{M_{\alpha,remote}}\log_2(1+\gamma)} \leq \frac{1}{\eta} \\ &\iff M_{\alpha,remote} \leq \frac{1}{\eta} \cdot \frac{1}{\frac{D_u}{B^u \cdot \log_2(1+\gamma)} + \frac{C}{Z_C} + \frac{D}{B^d \cdot \log_2(1+\gamma)}}, \end{aligned} \tag{5.1}$$

where η is the framerate, D_u the uplink data size, C the computational requirements and D the data size of the frame, B^u and B^d the up- and downlink bandwidths, γ the fixed SINR value and Z_C the total processing power at the MECs.

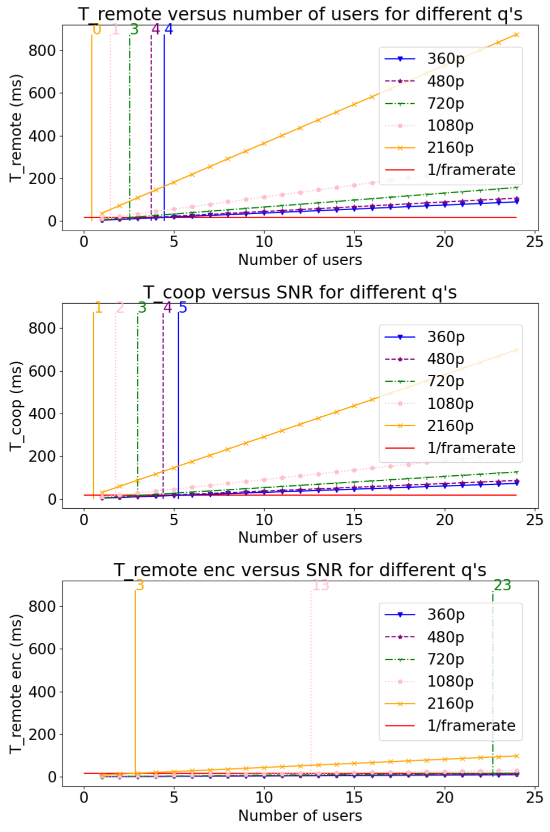


Figure 5.2: Offloading delays over α, the number of people at the base station, for $\gamma = 16.5$ dB. Vertical lines are intersections with $1/60$ s.

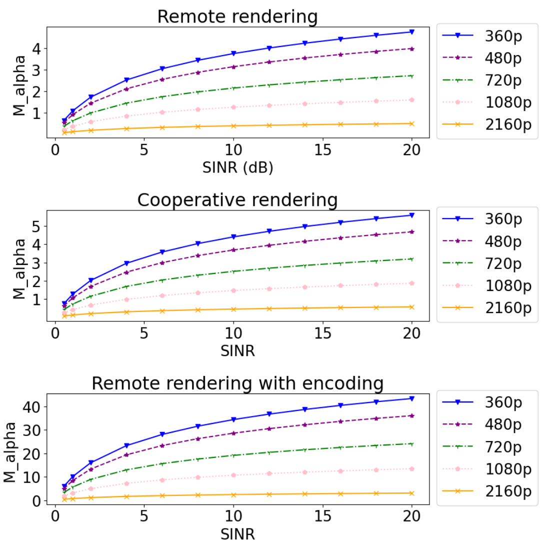


Figure 5.3: Maximum number of people that can be supported at the base station over SINR values for $B^u = 10$ MHz and $B^d = 150$ MHz.

For cooperative rendering as in Equation (4.7). Here, we assume that the local rendering part $C_f/z_l \leq \frac{1}{\eta} - t_{integrate}$, then:

$$T_{coop} \leq \frac{1}{\eta} \iff \max\left(\frac{C_f}{z_l}, \frac{C_b}{\frac{Z_C}{M_{\alpha,coop}}} + \frac{D_b}{\frac{B^d}{M_{\alpha,coop}} \log_2(1+\gamma)}\right) + t_{integrate} \leq \frac{1}{\eta}$$
$$\iff M_{\alpha,coop} \leq \left(1/\eta - t_{integrate} - \frac{C_b}{z_c}\right) \cdot B^d * \log_2(1+\gamma) \cdot D_b \tag{5.2}$$

where η is the desired framerate, D_u the data volume to transmit uplink, C the task size of the frame, D the data volume of the frame, B^u and B^d the up- and downlink bandwidths, γ the fixed SINR value and Z_C the total processing power at the MECs. Lastly, for remote rendering with encoding as in Equation (4.8):

$$T_{remoteenc} \leq \frac{1}{\eta} \iff \frac{D_u}{\frac{B^u}{M_{\alpha,remoteenc}} \log_2(1+\gamma)} + \frac{C}{\frac{Z_C}{M_{\alpha,remoteenc}}} + \frac{D}{\frac{Z_C}{M_{\alpha,remoteenc}}} \cdot b_{z_c}$$
$$+ \frac{D_k}{\frac{B^d}{M_{\alpha,remoteenc}} \log_2(1+\gamma)} + \frac{D_k}{z_l \cdot b_{zl}} \leq \frac{1}{\eta}$$
$$\iff M_{\alpha,remoteenc} \leq \left(\frac{1}{\eta} - \frac{D_k}{z_l \cdot b_{zl}}\right) \cdot \frac{1}{\frac{D_u}{B^u \log_2(1+\gamma)} + \frac{C}{Z_C} + \frac{D}{Z_C \cdot b_{z_c}} + \frac{D_k}{B^d \log_2(1+\gamma)}}, \tag{5.3}$$

where η is the desired framerate, D_u the data volume to transmit uplink, C the task size of the frame, D the data volume of the frame, B^u and B^d the up- and downlink bandwidths, γ the fixed SINR value and Z_C the total processing power at the MECs.

Figure 5.3 plots these equations against different SINR values for $B = 150\text{MHz}$. The results show that with increasing SINR, more users can be supported, but for remote and cooperative rendering, the numbers still stay low, below 10. For remote encoding the $M_{\alpha,remoteenc}$ are reasonably high for low resolutions, but still only 3 for rendering frames at 2160p (vertical line in Figure 5.2).

Table 5.1: Example offloading delays in seconds when $B = 150$ MHz and $\gamma = 16.5$ dB. $T_{remoteenc}$ assumes the HMD needs to decode the entire file size.

Resolution q	T_{local} **(ms)**	T_{remote} **(ms)**	T_{coop} **(ms)**	$T_{remoteenc}$ **(ms)**
360	5.76	3.73	3.98	0.96
480	6.88	4.45	4.56	1.15
720	10.08	6.52	6.22	1.68
1080	17.28	11.18	9.94	2.88
2160	56.16	36.33	30.06	9.36

Finally, what typical offloading delays can we expect for reasonable SINR and bandwidth? Table 5.1 summarises the offloading delays under *good* network conditions ($\gamma = 16.5\text{dB}$) and a bandwidth of $B = 150\text{MHz}$.

Chapter 6

Proposed Offloading Heuristics

This chapter describes and proposes the offloading heuristics GREEDY and OPP-BUFFER, as well as some simple strategies that act as baselines to investigate the research questions. Table 6.1 summarises the main idea, variants and parameters of the different rendering strategies. A rendering strategy consists of two parts: what offloading path (0,1,2,3) and what quality resolution to choose. The possible resolutions are 360p, 480p, 720p, 1080p or 2160p and the possible offloading paths are:

0: Local computations;

1: Remote computations without encoding and decoding;

2: Cooperative computations without encoding and decoding;

3: Remote computations with encoding and decoding.

Table 6.1: Summary of the main idea, the variants and the parameters of the proposed heuristics.

Strategy	Idea	Variants	Parameter
GREEDY	Highest resolution possible under current network circumstances, default computing locally at 720p	REMOTE, REMOTE ENC	SINR perturbation factor σ
OPP-BUFFER	Highest resolution possible if enough frames are buffered, default resolution q^*, always the same offloading path o.	LOCAL, REMOTE, REMOTE-ENC	Per variant: default q^* and expected rendering delays t_q's.
O-Q	Static rendering strategy, always choose the same offloading mode o and resolution q	All possible combinations	Offloading path o and resolution quality q.
RANDOM	Random offloading mode $o \in \{0, 1, 2, 3\}$ and random quality resolution $q \in \{360, 480, 720, 1080, 2160\}$	If the resolution is fixed: RANDOM-<RESOLUTION>	-

6.1 Base-line offloading strategies

Static rendering strategies

The simplest base case is an offloading strategy that always chooses the same offloading path and the same quality resolution. Naming-wise they are designated by <OFFLOADING PATH> - <RESOLUTION>

We consider as baselines the strategies that always choose the same offloading path but do not risk stalling. Ergo, the resolution is chosen such that at $B = 150$ MHz, the offloading delay is below $1/\eta = 16.7$ ms for average SINR. As shown in Figure 5.1b, this holds for the following: LOCAL-720, REMOTE-1080, REMOTE-ENC-2160 and COOP-1080.

Random choice

To evaluate whether designing an offloading strategy even makes sense, we also consider a *random* offloading strategy that chooses an offloading path at random as well as a resolution at random. If the randomness is limited to the offloading decision, the rendering strategy is denoted by RANDOM-<RESOLUTION> and does not alter the chosen resolution quality.

6.2 Heuristic offloading strategies

To answer RQ 1, we devised two different offloading heuristics that dynamically base their rendering decision on a different aspect: GREEDY calculates the expected rendering delay based on current channel condition estimates, and OPP-BUFFER bases its decision on the current buffer state. The primary goal of both heuristics is to minimise the risk of system stalls at the desired framerate of η FPS. This risk is eliminated if a frame is rendered under $1/\eta$ s. The secondary goal of the heuristics is to render the frames at the highest quality possible. Moreover, we have designed explainable heuristics with low computational complexity. Both GREEDY and OPP-BUFFER have a complexity of $O(n_q)$, where n_q is the number of resolutions that both heuristics attempt to use. In our setting, $n_q \leq 5$, since we consider 5 different quality resolutions that are possible.

6.2.1 GREEDY: highest quality based on estimated channel quality

Algorithm 1: GREEDY rendering heuristic G-X.

```
Data: Favoured offloading path $X \in [0, 1, 2, 3]$, state information $S$ (available bandwidth,
      processing power, etc.), SINR perturbation factor $\sigma$, framerate $\eta$.
Result: The frame's rendering decision $\phi = (o, q)$ with the selected offloading path $o$ and
        resolution quality $q$.
$q_s \leftarrow [2160, 1080, 720]$ ;                    /* List of quality resolutions */
$o \leftarrow X$ ;                  /* G-X where X is the favoured offloading path */
/* Starting with the highest quality, check whether the estimated
   delay is below 1/framerate                                                 */
for $q \in q_s$ do
    $t$ = get_estimated_rendering_delay_with_quality($q, o, S, \sigma$ )
    if $t \leq 1/\eta$: then
        ;   /* If delay lower than threshold, offload and stop loop.  */
        return $(o, q)$
    end
    /* else consider the next largest resolution                              */
end
/* Default is local offloading at 720p, which is below $1/\eta$               */
return $(0, 720)$
```

The GREEDY heuristic has three variants, denoted by G-REMOTE, G-REMOTE-ENC and G-COOP, but we concentrate on the two former variants. The idea behind GREEDY is to attempt to offload the rendering of a frame at the highest resolution possible under the current network circumstances. The pseudocode in Algorithm 1 starts with the highest resolution and calculates the corresponding offloading delay. G-REMOTE calculates T_{remote} as in Equation (4.6), G-REMOTE-ENC calculates $T_{remoteenc}$ as in Equation (4.8) and G-COOP calculates T_{coop} as in Equation (4.7) under the current system state.

If the delay is below $1/\eta$ s, then GREEDY chooses to render at the MECs (with or without encoding, respectively). If it is above, the algorithm repeats the calculations with a resolution of 1080p. If this estimated delay is below $1/\eta$ s, it chooses remote offloading at 1080p. Otherwise, it tries to offload at 720p. If the estimated offloading delay at 720p is also larger than $1/\eta$ s, then it chooses its default offloading option, namely to render *locally* at 720p.

An advantage of this heuristic is that it attempts to render the highest resolution possible under current network circumstances. Furthermore, the default alternative of locally rendering at 720p will always work, as shown in Chapter 5. Finally, this heuristic is not dependent on the buffer, so it can also be used in a real-time rendering scenario where frames are displayed as soon as they are rendered. A disadvantage is that this heuristic requires estimates of all values in Equations 4.6 or 4.8. Even getting the correct system parameters is a challenge in itself that would have to be tackled before implementing this heuristic in practice. We do not face this complication in the simulation environment since we define all system parameters at the start. To estimate the channel conditions or, more precisely, the SINR values, the simulator takes the actual SINR value γ that is simulated and perturbs it with a factor σ. Then, GREEDY calculates not with the actual SINR value γ but with an over- or underestimation of $\gamma \pm \sigma \cdot \gamma$ instead. For this work, by default $\sigma = 1$ to see the full potential of GREEDY.

6.2.2 OPP-BUFFER: highest quality based on buffered frames

Algorithm 2: OPP-BUFFER rendering heuristic OB-X.

```
Data: Favoured offloading path $X \in [0, 1, 2, 3]$, state information $S$ (available bandwidth,
      processing power, etc.), OPP-BUFFER parameters $q^*$, $t_{2160}$ and $t_{1080}$, framerate $\eta$, buffer.
Result: The frame's rendering decision $\phi = (o, q)$ with the selected offloading path $o$ and
        resolution quality $q$.
$t_{budget} \leftarrow$ length(buffer) $\cdot\ 1/\eta$ ;      /* Determine the already buffered video
 duration */
$o \leftarrow X$ ;            /* OB-X where X is the favoured offloading path */
/* Choose the first resolution whose pre-determined parameter fits
   into this budget                                                 */
/* Start from the highest resolution                                */
if $t_{budget} > t_{2160}$: then
    ;      /* If delay lower than threshold, offload and stop loop.  */
    return $(o, 2160)$
else
    if $t_{budget} > t_{1080}$: then
        ;   /* If delay lower than threshold, offload and stop loop.  */
        return $(o, 1080)$
    end
end
/* Continue until default resolution $q^*$.                         */
return $(o, q^*)$
```

For local, remote and remote encoding, we denote the respective OPP-BUFFER strategy by OB-LOCAL, OB-REMOTE and OB-REMOTE-ENC, respectively. The main idea behind OPP-BUFFER is to render frames at higher resolutions only if the buffer contains enough frames to display to the user until the higher resolution frame is rendered. As shown in the pseudocode in Algorithm 2, OPP-BUFFER first calculates a *budget*, namely how much time it maximally has to render a frame. The budget is calculated by

Table 6.2: Parameters for OPP-BUFFER heuristics

OPP-BUFFER	q^* (p)	t_{2160} (s)	t_{1080} (s)
OB-LOCAL	720	0.057	0.0173
OB-REMOTE	1080	0.0273	0.0084
OB-REMOTE-ENC	2160	0.0084	0.0026

multiplying the number of frames in the buffer by the desired inter-frame rate $1/\eta$. If the budget is high enough to render a frame at 2160p, so higher than t_{2160}, the heuristic chooses to render at 2160p. If not, it checks whether the budget allows for 1080p and so forth until a pre-determined value q^*, which is the default resolution if all higher resolutions are not possible. The X in the Algorithm 2 stands for the respective offloading path because OB-X always makes the same offloading choice, namely $o = X$. This implies that all parameters q^* and t_q's vary between the OPP-BUFFER variants. Table 6.2 summarises the parameters chosen for every variant. The reasoning for these parameter values originates from the analytical analysis in Chapter 5. The q^* value are chosen in such a way that for $B = 150$ MHz under *good* conditions, rendering at q^* is below $1/\eta$. The delay numbers for t_q's originate from Table 5.1.

When designing OPP-BUFFER, we have chosen to hard-code the t_q values to avoid GREEDY's disadvantage of needing complete knowledge of the system state. Therefore, an advantage of OPP-BUFFER is that it only requires the number of currently buffered frames. However, the main advantage of OPP-BUFFER is that it can render frames at resolutions with "too large" offloading delays by using the buffer space. The main disadvantage is that this forbids us from using the heuristic in a real-time setting where no buffer is kept. A second disadvantage is that the parameters are indeed fixed and cannot adapt to the actual system state.

Chapter 7

Simulation Framework

This chapter presents the simulation framework developed to answer the research questions. The simulator simulates a VR game where the HMD can make rendering decisions based on the current system state and optionally offload to MEC servers. To incorporate the streaming-like aspect of VR games, it simultaneously simulates the rendering and the display process and evaluates the quality of experience as a whole. This simulator is used in the performance evaluation of Chapter 8.

The core of the event-based simulation framework consists of a global system state, a simple loop and a sorted event queue. At every iteration, the *next* event is taken from the queue and executed. An event can, in turn, alter the system state and add new events to the queue. The queue is ordered according to the virtual time in the simulation. Progressing the simulation time happens when executing an event. Then, the current simulation time is set to the time of the event. When adding new events to the queue, we must calculate when to execute them and sort them in the queue accordingly. This procedure repeats until no events are left in the queue or a specific ending criterion triggers.

In essence, the events and their interactions need to simulate the play-through of a VR game. The simulator achieves this by translating the play-through of a VR game into discrete steps. As shown in Figure 7.1, we devised a *simulation flow*, where playing a VR game (the *playing phase*) consists actually out of two different loops, the *display loop* and the *rendering loop*. The *display loop* can be divided into consecutive "Display next frame" events. Since we assume a fixed playback model at η frames per second, every $1/\eta$, the system should display a frame. The *rendering loop* consists of the alteration between two events, "Render next frame", so start the rendering process, and "Rendered frame arrived at HMD", meaning that the ready-to-display frame is at the HMD. The time between both events depends on the offloading delay, which is determined by the chosen rendering strategy. As typical with video applications, the rendering loop begins earlier than the display loop to build up a buffer already and improve the replay experience. This process is called the "Start-up phase" and usually ends as soon as a fixed number of frames is computed.

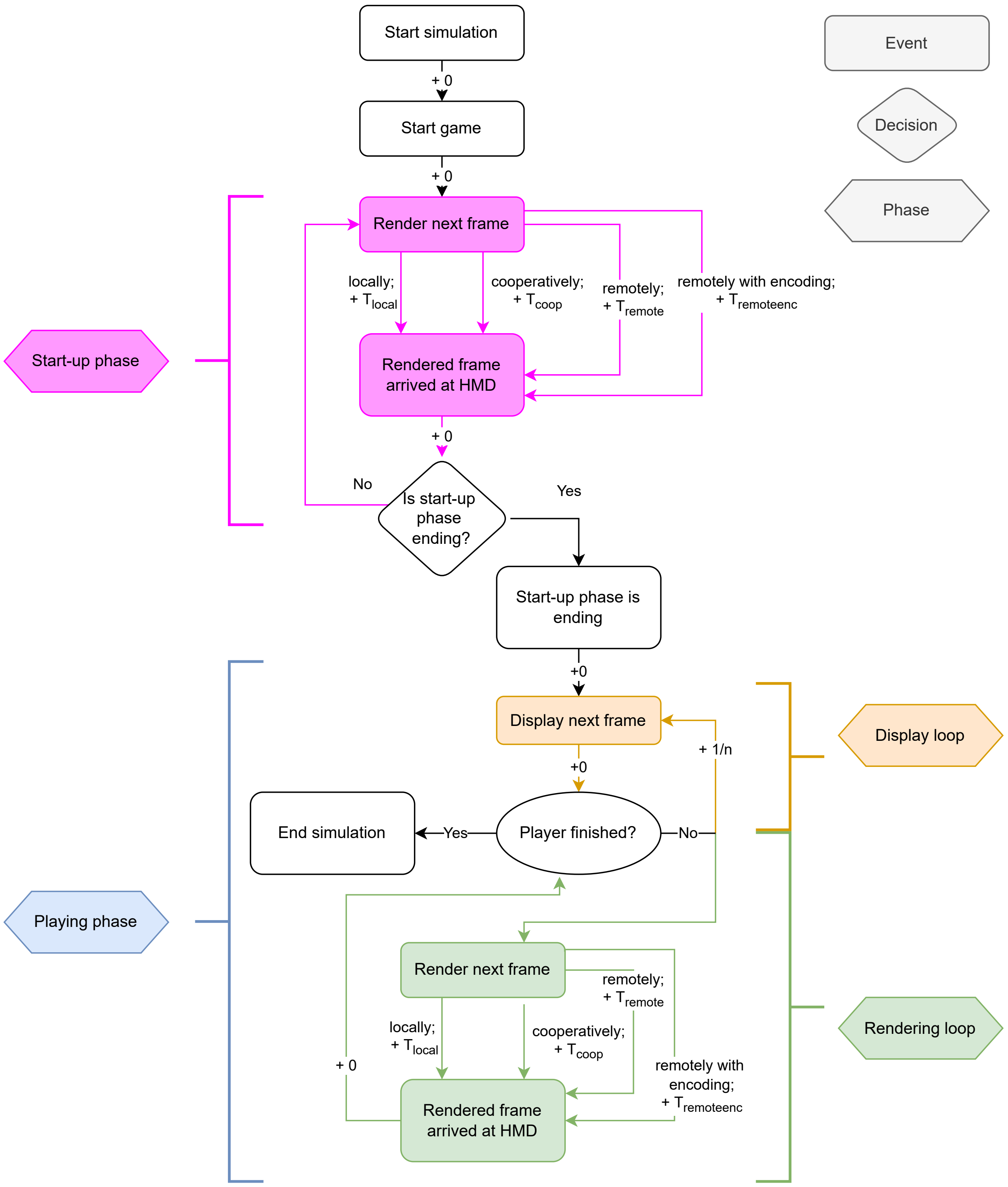


Figure 7.1: Overview of the simulation flow.

The VR experience is simulated in our framework using the following events, which will be explained one-by-one below:

- Start Game event in Figure 7.2;
- Start-up phase ending event in Figure 7.3;

- Render next frame event in Figure 7.4;
- Rendered frame at HMD event in Figure 7.5;
- Display next frame event in Figure 7.6;
- End simulation event in Figure 7.7.

In all the simulation flow figures, the magenta-coloured boxes and text do not aid with the simulation process but explain how the performance metrics in Section 8.1 are calculated.

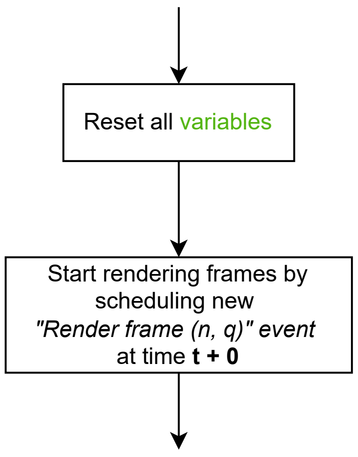

Reset all variables
Start rendering frames by scheduling new *"Render frame (n, q)" event* at time **t + 0**

Figure 7.2: Event *"Start game"*

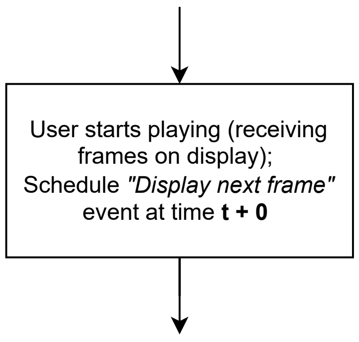

User starts playing (receiving frames on display); Schedule *"Display next frame"* event at time **t + 0**

Figure 7.3: Event *"Start-up phase ending"*

The *"Start game"* event in Figure 7.2 starts the simulation by scheduling the first "Render next frame" event. When a user starts playing a game, we *assume* that the HMD immediately starts rendering the required frames. However, the player sees a loading screen until the start-up phase is finished. The start-up phase is finished once a pre-defined number of frames b has been rendered and stored in the buffer. Once the start-up phase is supposed to end, a *"Start-up phase ending"* event is scheduled, which commences the display loop by scheduling a new "Display next frame" event.

The *"Render next frame"* event takes into account the chosen offloading mode o and quality resolution q. Depending on o, the "Rendered frame at HMD" is scheduled after the corresponding offloading delay. The offloading delay is calculated according to the system state at time t and the computing model in Section 4.4.

The events *"Rendered frame at HMD"* and *"Display next frame"* in Figure 7.5 and Figure 7.6 contain the core functioning of the VR game simulation. When a rendered frame arrives at the HMD, the simulation framework first has to check whether the start-up phase has already ended. If not, it must check whether the arrival of the new frame ends the start-up period. During start-up, the buffer space can never be full (otherwise, the start-up has already ended), but afterwards, the simulator must check whether there is space. If the buffer is full, the newly arrived frame must be dropped, and the simulation halts the rendering loop until a new frame is displayed (during the *"Display next frame"* event). Otherwise, the arriving frame is added to the buffer. Then, the system needs to check whether the game is in a *stall*, so whether the display loop was stopped because the buffer was empty. If t_{stall} is set, this has been the case, and the newly arrived frame can resolve the stall. At this moment, the display loop restarts by scheduling a new *"Display next frame"* event immediately after the resolution of this event, as there is now a frame to display. In the end, if the system is not in a stall or the start-up phase, the event ensures that a new frame is rendered immediately after by scheduling a *"Render next frame"* event.

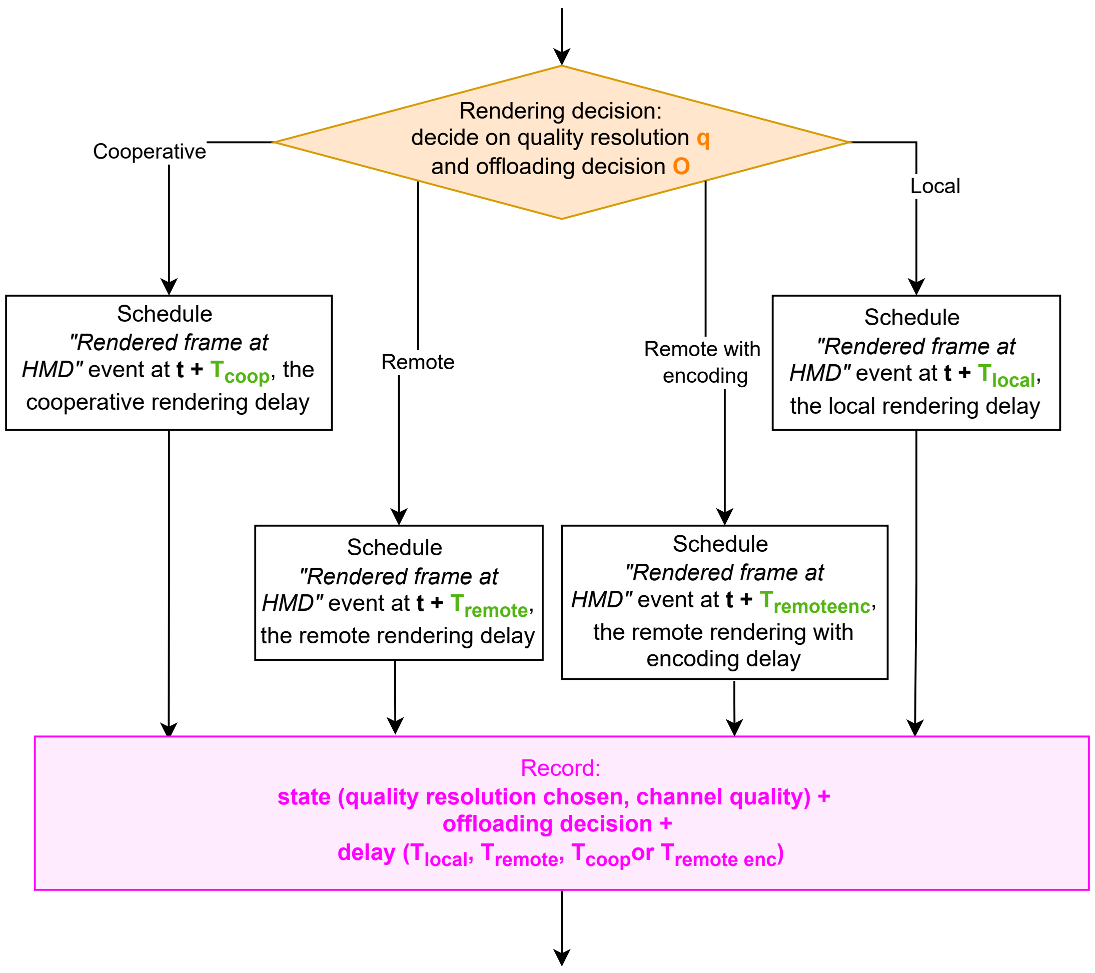

Rendering decision: decide on quality resolution q and offloading decision O
Cooperative
Remote
Remote with encoding
Local
Schedule "Rendered frame at HMD" event at t + T_coop, the cooperative rendering delay
Schedule "Rendered frame at HMD" event at t + T_remote, the remote rendering delay
Schedule "Rendered frame at HMD" event at t + T_remoteenc, the remote rendering with encoding delay
Schedule "Rendered frame at HMD" event at t + T_local, the local rendering delay
Record: state (quality resolution chosen, channel quality) + offloading decision + delay (T_local, T_remote, T_coop or T_remote enc)

Figure 7.4: Event *"Render next frame"*

In the display loop, the HMD displays rendered frames to the user in the desired framerate via its display. When a new frame should be displayed, the system needs to check whether the buffer contains a rendered frame. If this is not the case, the system stalls, and we must wait for a rendered frame to arrive at the HMD. Furthermore, the simulator must check if the rendering loop stopped due to a full buffer. If so, the simulator needs to restart the rendering loop by scheduling a new "Render next frame" event. Only then the oldest frame is shown to the user and taken from the buffer. We still assume that the FoV prediction is always correct, but in future work, the simulator could be extended to include wrong predictions and potential re-computations. We schedule an "End simulation" event if the player is finished. Otherwise, we schedule a new "Display next frame" event at the interval corresponding to the desired framerate η. In our simulations, the user stops playing after a fixed time T.

At the end of the simulation, we calculate the necessary statistics and export the values to a .csv file, as shown in Figure 7.7.

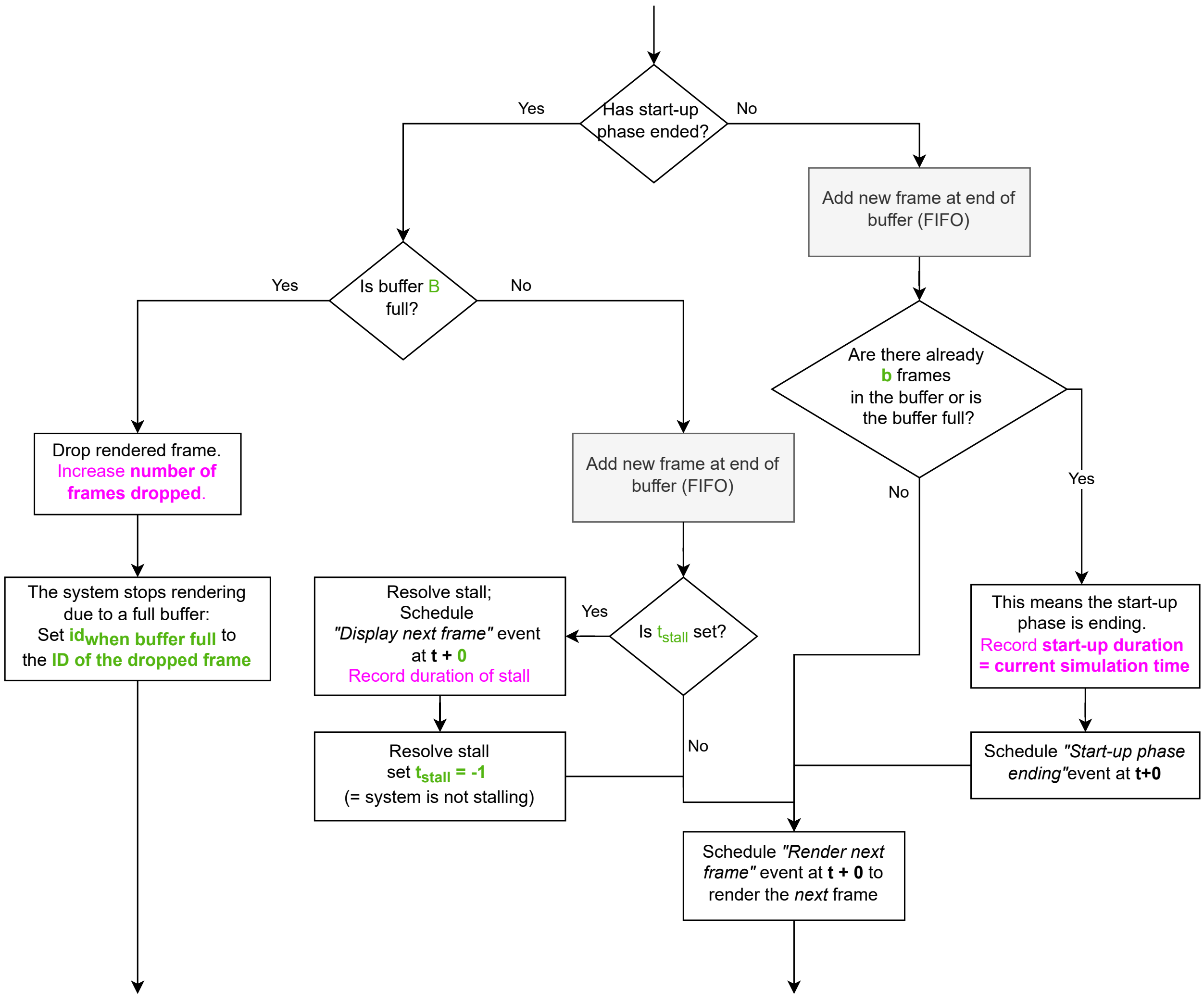

Has start-up phase ended?
Yes
No
Add new frame at end of buffer (FIFO)
Is buffer B full?
Yes
No
Are there already b frames in the buffer or is the buffer full?
Drop rendered frame. Increase number of frames dropped.
Add new frame at end of buffer (FIFO)
No
Yes
The system stops rendering due to a full buffer: Set id_when buffer full to the ID of the dropped frame
Resolve stall; Schedule "Display next frame" event at t + 0 Record duration of stall
Yes
Is t_stall set?
This means the start-up phase is ending. Record start-up duration = current simulation time
Resolve stall set t_stall = -1 (= system is not stalling)
No
Schedule "Start-up phase ending" event at t+0
Schedule "Render next frame" event at t + 0 to render the next frame

Figure 7.5: Event *"Rendered frame at HMD"*

Modular simulation framework: pick your inputs

Our goal is to keep the same core simulation structure in different simulation settings. Hence, different rendering strategies, stopping methods, display methods, signal-to-noise generators and other system parameters can be passed as input arguments to the simulation framework. In the following paragraphs, we elaborate on these concepts. Any of the *rendering strategies* presented in Chapter 6 is implemented in the framework and can be selected for a simulation run. At the "Render next frame" event, the chosen rendering strategy determines the offloading path and the resolution quality. Within the framework, a rendering strategy has access to all system components, even if they might not be needed. The *stopping strategy* indicates how long the simulation is running. Currently, we only implemented the most straightforward stopping strategy, namely stopping after some fixed time T. Similarly, the simulation framework supports different *display strategies* that, after displaying a frame, provide the number of seconds after which to display the next frame. Again, the simulator implements the most straightforward strategy: displaying frames at constant framerate η FPS.

A source of uncertainty in the system is the experienced signal-to-interference-and-noise ratio (SINR) of the HMD. At every time step, the simulator provides SINR values according to some *SINR*

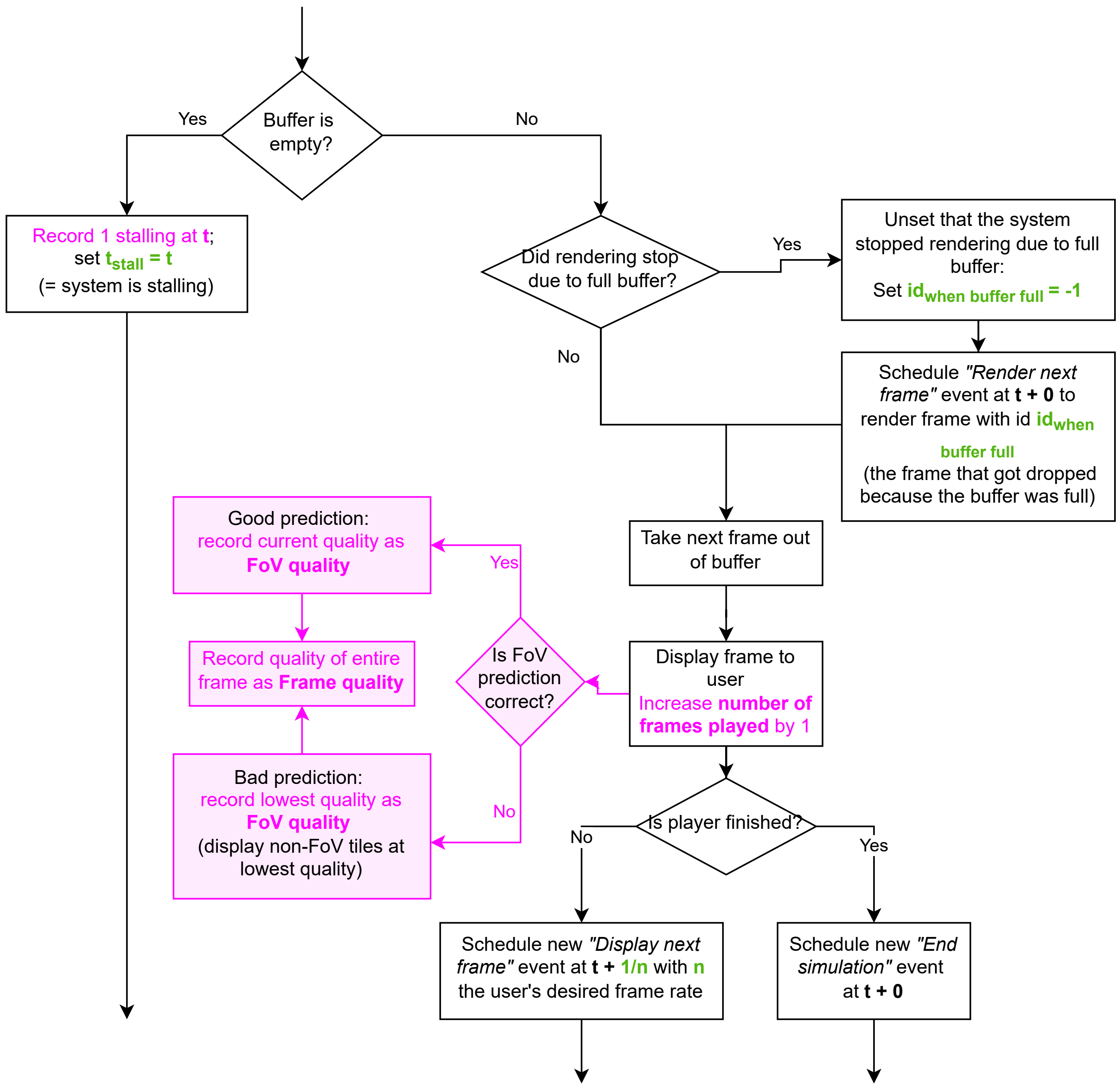

Buffer is empty?
Yes
No
Record 1 stalling at t; set t_{stall} = t (= system is stalling)
Did rendering stop due to full buffer?
Yes
No
Unset that the system stopped rendering due to full buffer: Set $id_{when\ buffer\ full}$ = -1
Schedule "Render next frame" event at t + 0 to render frame with id $id_{when\ buffer\ full}$ (the frame that got dropped because the buffer was full)
Take next frame out of buffer
Display frame to user Increase number of frames played by 1
Is FoV prediction correct?
Yes
No
Good prediction: record current quality as FoV quality
Record quality of entire frame as Frame quality
Bad prediction: record lowest quality as FoV quality (display non-FoV tiles at lowest quality)
Is player finished?
No
Yes
Schedule new "Display next frame" event at t + 1/n with n the user's desired frame rate
Schedule new "End simulation" event at t + 0

Figure 7.6: Event *"Display next frame"*

generator. In the current simulator version, the default SINR generator draws SINR values according to a uniform distribution between 1 dB (close to the cellular edge) and 20dB (excellent connectivity). This implies that the individual SINR values are independent (of the time and previous values) and identically distributed. The simulator can be extended with more complex SINR value generators. For instance, drawing SINR values according to a time-dependent distribution, using existing values from a file or trace [28] or calculating them based on the current state (position, noise, interference and transmission power). In any case, if a random number generator is used, the used seeds are saved to a file for reproducibility.

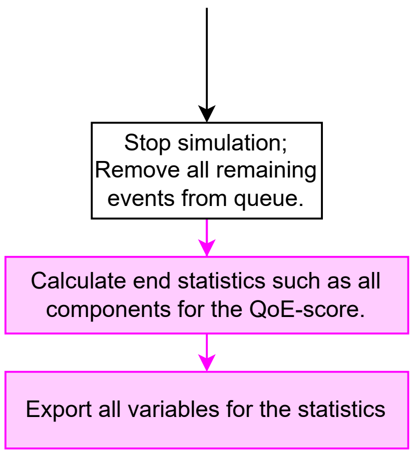

Stop simulation; Remove all remaining events from queue.
Calculate end statistics such as all components for the QoE-score.
Export all variables for the statistics

Figure 7.7: Event *"End simulation"*

Chapter 8

Performance Evaluation

To answer RQ 1 and RQ 2, the heuristics in Chapter 6 are evaluated using the proposed simulation framework in Chapter 7. This chapter is organised as follows. First, Section 8.1 describes the simulation set-up, including the simulated scenarios, the measured performance metrics and the system parameters. Then, Section 8.2 presents the performance evaluation results. Following the structure of RQ 2, we answer the following questions in sequence:

2a What is the impact of the downlink bandwidth on the QoE-score of rendering strategies?

2b What is the impact of the maximum buffer capacity on the QoE-score of rendering strategies?

2c What is the impact of the channel quality on the QoE-score of rendering strategies?

2d What is the impact of the compression ratio on the QoE-score of strategies that use remote rendering with encoding?

During the above, we also closely compare the different rendering strategies to answer RQ 1.

8.1 Simulation set-up

The simulator of Chapter 7 has been implemented using PYTHON. All simulation experiments have been conducted on a Lenovo P20 laptop with an Intel(R) Core(TM) i7-6700HQ processor running Windows 10. The code can be found under "VR Gaming Offloading Decision Simulation" at the University of Twente's GitLab [30]. The git-repository includes the .csv files with the simulation results and the random number generator's text files with the seeds. The simulated setting is described in Chapter 4. Frames can be computed locally or remotely at the MECs using pre-stored mathematical models to render the required image. The system is equipped with a buffer space that can hold M_B frames into which ready-frames are stored until they are displayed to the user's HMD display at a fixed framerate of η FPS. For every frame, the proposed rendering heuristic determines the offloading mode o and the resolution q at which the frame should be rendered. The primary performance metric is the QoE-score defined in Equation (4.18).

While other base-line scenarios have also been simulated, we only included LOCAL-720, REMOTE-1080 and RANDOM-1080 in the comparison against the heuristics GREEDY and OPP-BUFFER. We chose LOCAL-720 because it does not make use of offloading (shedding light on the question of whether offloading is indeed advantageous), and REMOTE-1080 because it does, but not smartly. Chapter 5 shows that 720p and 1080p are the largest resolutions that can be computed locally and remotely under the threshold of 1/60 s. RANDOM-1080 was included to see whether designing dedicated rendering strategies merits attention at all.

Table 8.1: Supported performance metrics.

Name	Description	Unit
Number of stalls	The number of times the buffer is empty when next frame should be displayed.	stalls
Total duration stalls	The sum of every stall's duration.	s
Number of dropped frames	The number of times the buffer is full when a rendered frame arrives at the HMD.	frames
Number of frames displayed	The total number of frames shown to the user.	frames
Average FoV quality of a frame	The average resolution quality of all *displayed* frames.	p
Number of decisions	Counts the total number of rendering decisions made and equals the number of rendered frames. It can exceed the number of displayed frames.	decisions
Decision frequencies	How often every offloading path is chosen.	-
Average rendering delays	The rendering delays are calculated per offloading path according to Equations (4.5)-(4.8).	s
Number of quality changes	How often the quality between two consecutive frames changed.	-
Average quality change	The average difference between two consecutive frames' data sizes.	bits
Average age of a frame	The average time between starting the rendering process and eventually displaying the frame.	s
Start-up duration	The time until the start-up phase ended, meaning that enough frames were added to the buffer.	s
Total simulation time	The time at which the simulation stopped.	s
Per offloading path: QoE utility	Guo et al.'s main performance metric counting how often the rendering delay is below $1/\eta$ [16].	-
Average buffer length on display	How many frames are left in the buffer on average after displaying the next frame.	frames
Average buffer length when rendered frames arrive at HMD	How many frames are already in the buffer at the arrival of the new frame.	frames
Achieved frames per second	Equals the number of frames displayed divided by the total simulation time.	fps
Achieved frames per second without start-up delay	Equals the number of frames displayed divided by the difference between the total simulation time and the start-up duration.	fps
Average stall duration	Calculated by dividing the total duration of stalls by the total number of stalls.	s
Offloading decision percentages	Per offloading path: divide the number of times each offloading path is chosen by the total number of decisions and multiply by 100.	%
Percentage of frames that incurred a quality change	Calculated by dividing the number of quality changes between consecutive frames by the number of displayed frames times 100.	%
Percentage of rendered frames that got displayed	Calculated by the division between the number of displayed frames and the number of rendered frames times 100. The non-displayed frames were either dropped or still in the buffer when the simulation stopped.	%

Performance metrics (y axes)

To calculate the QoE-scores defined in Section 4.6, the simulator keeps track of several metrics during the simulation. Next to all information needed for QoE-score, the simulator also tracks the performance metrics described in Table 8.1.

Statistical significance

To ensure the statistical significance of the above metrics, every simulation scenario is run 50 separate times with different seeds. Uncertainty originates from the chosen values for the channel quality which are determined by SINR generators. In our performance evaluation, SINR values are drawn uniformly between 1 and 20 dB except otherwise mentioned. This implies that individual SINR values are independent and identically distributed. Since these SINR generators are random number generators, we choose a new seed for every run and save it to a file for reproducibility. In this way, every run experiences identically distributed SINR values (as the only source of uncertainty), implying that the performance metrics are also identically distributed. Furthermore, running entirely separate occasions (i.e. restarting the simulation every time) ensures the independence between the different runs' performance values. Lastly, 50 is a large enough number that we can apply a law for large numbers.

Putting all of the above together, we can apply the Central Limit Theorem to calculate 95%-confidence intervals with z-values. The computed intervals are plotted together with the performance metrics in Section 8.2 but are largely invisible due to small widths. As a consequence, we are 95%-confident that, on average, the performance metric's mean is indeed as depicted in the figures.

Fixed parameters

Unless otherwise specified, we use the parameters defined in Table 8.2. This section provides arguments for the various parameter choices. First of all, the desired framerate η is set to 60 FPS, commonly referred to as the minimal framerate needed for a satisfying VR experience [16]. Oculus Rift, a state-of-the-art VR HMD, also supports 60 FPS [8]. The colour information of a single pixel is set to 10 bits according to Snapdragon 835, commonly used in VR [36], which provides a deeper colour than its 8-bit alternative. According to Hooft et al., many tiling schemes are possible, so we chose for $l \times w = 8 \times 6$ as in the Sandwich and Spotlight 360° video [17]. Since the human vision has a field of view of 110° [14], which is supported by the Oculus Rift specs [8], we calculated that 3×4 tiles cover the FoV at all times.s Many system parameters are based on the model specifications of Guo et al. [16], including the processing powers Z_C and z_l at MECs and HMD, as well as the number of bits that can be computed in a CPU cycle at both destinations b_{zl} and b_{zc}. Furthermore, the uplink and downlink bandwidths by Guo et al., namely 10 MHz and 1 GHz, have been a starting point. We set the uplink bandwidth to 10 MHz but reduce the downlink bandwidth to 150 MHz. This choice is because Guo et al. assume that approximately ten users share the same bandwidth [16] while we assume a single user is connected to the BS. Furthermore, a downlink bandwidth of 150 MHz offers data rates close to 1 Gbps, the promised speed by 5G networks. The empirical experiments by Lai et al. [18] are the basis for choosing an integration delay $t_{integrate} = 0.001$s. The compression coefficient k is set to 10, larger than the coefficient assumed by Guo et al. [16] which was set to 5, but lower than the reduction of almost 35 measured by Lai et al.[18]. Hence, we deem the coefficient reasonable but not overly ambitious since compression highly depends on the encoding algorithm and the video content. Moreover, we assume a small buffer size M_B of 10 frames and compute $b = 3$ frames in the start-up phase. Finally, the ratio between background and foreground β is set to 0.8 to reflect the assumption that the background is richer and more detailed than the foreground. Finally SINR values are drawn uniformly between 1 and 20 dB, representing channel qualities ranging from *bad* (close to 1 dB), up to *excellent* (close to 20 dB).

Table 8.2: Table of simulation parameters unless otherwise specified.

Symbol	Name	Value	Unit
η	Desired framerate	60	FPS
$t_{integrate}$	Delay to integrate back- and foreground of a frame on the HMD	1	ms
k	Encoding data ratio between raw and encoded frame	$1/10$	-
-	Encoding coefficient	1	-
-	Decoding coefficient	=k=1/10	-
b_{z_c}, b_{z_l}	Number of bits processed in a CPU cycle at MECs or locally	0.4	bits/CPU cycle
-	Bits per pixel to store colour information	10	bits
$l \times w$	Tiling scheme: frame divided into $l \times w$ tiles in length and width	8×6	tiles
n	Number of tiles in the FoV	4×3	tiles
-	Aspect ratio of a frame	16:9	-
-	Uplink data ratio (ratio that needs to be communicated uplink)	1/1000	-
β	Proportion of a frame corresponding to the background	0.8	-
Z_C	Total processing power of the MECs	1000e9	Hz
z_l	Local processing power of the HMD	1e9	Hz
α	Number of people associated to the base station	1	user
B^u	Uplink bandwidth	10	MHz
B^d	Downlink bandwidth	150	MHz
γ	Signal-to-noise-and-interference-ratio, drawn from uniform distribution	$U(1, 20)$	dB
Θ	Maximally achievable bitrate of VR game-play	Mbits/s	1348
M_B	Maximal buffer size	10	frames
b	Number of frames to buffer before displaying first frame	3	frames
κ_{1-4}	Coefficients of the QoE-score	1	-

Simulation scenarios based on variable system parameters

To answer RQ 2 and investigate the influence of system parameters, one by one, we run simulations for a varying parameter space for:

1. **Up- and downlink bandwidth B^u and B^d**: As all offloading paths except local rendering use network transmissions, it is essential to vary network parameters such as the bandwidth. The bandwidths range between 50MHz and 950 MHz. According to Shannon's formula, a bandwidth of 150MHz under excellent conditions results in a data rate of approximately 1 Gbps. 5G promises such data rates to its users, making the emergence of applications such as VR possible. However, we would like to examine the influence of even better data rates in beyond 5G networks. Therefore we chose a range going up until 950 MHz. The upper value stems from Guo et al.'s experiments, which assume an uplink bandwidth of 1 GHz for 100 users and 10 BSs. Note that the simulated user is the only user present and can use the entire bandwidth and MECs' processing power for themselves.

2. **SINR distributions**: In default scenarios, the SINR value γ is drawn uniformly between 1 and

20 dB. In these runs, we vary the interval from which to uniformly draw the SINR values from low values ($[1, 2]$ and $[2, 3]$ dB) to larger intervals (until $[10, 15]$ and $[15, 20]$ dB).

3. **The maximum buffer capacity M_B**: Buffering frames might significantly impact the user's overall experience, so it is relevant to evaluate the rendering strategies' performance on a varying maximum buffer capacity. We simulate M_B from 1 to 30. Buffer capacities close to 1 resemble scenarios where the rendering has to be done in real-time since not many frames can be buffered, while buffer capacities around 30 can store up to 0.5s of game content.

4. **The encoding ratio k**: this scenario is limited to heuristics that render remotely with encoding. This coefficient determines by what ratio the compressed file-sizes decreases compared to the original file for downlink transmission and decoding. We evaluate k, ranging between 1 and 45.

8.2 Simulation results for a VR game with buffer space

This section first presents the impact of the bandwidth on the rendering process in Section 8.2.1, then the impact of the maximum buffer capacity in Section 8.2.2, afterwards the impact of the SINR in Section 8.2.3 and finally the impact of the encoding coefficient in Section 8.2.4 to answer RQ 2. Furthermore, all above sections have as second goal to evaluate the performance of the proposed heuristics GREEDY and OPP-BUFFER against constant base cases to answer RQ 1: Do *dynamic* rendering strategies outperform *static* rendering strategies in terms of QoE-score? Note that all the values in the figures are averages over 50 runs and confidence intervals are plotted (although invisible).

8.2.1 Impact of the up- and downlink bandwidth

The main result of this section is depicted in Figure 8.1 showing the QoE-score of all evaluated rendering strategies over (up- and downlink) bandwidths ranging from 50 MHz to 950 MHz. The structure of the section is organised as follows: First, we observe the impact of the bandwidth in general, then we compare the individual rendering strategies and afterwards we discuss potential explanations for the observations using other performance metrics.

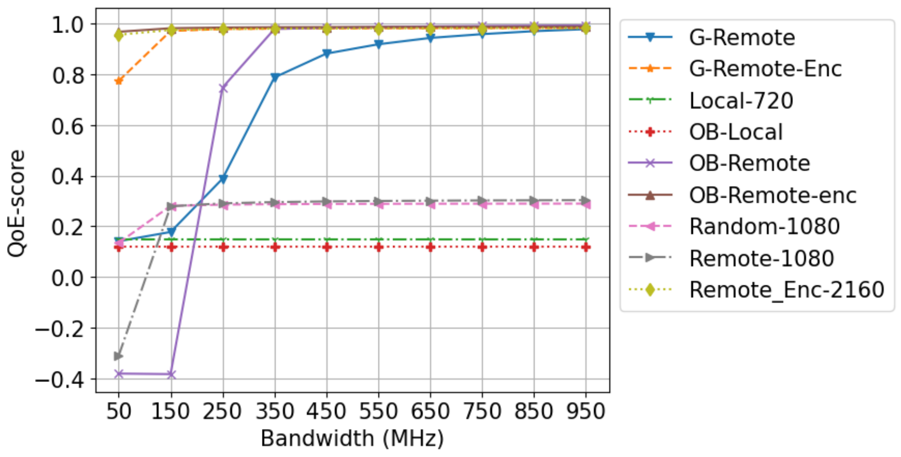
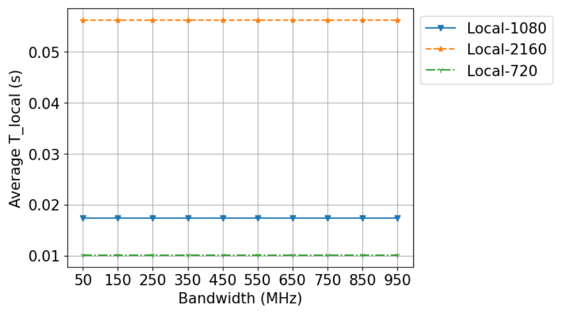


Figure 8.1: QoE score over the up- and downlink bandwidths B^u and B^d.

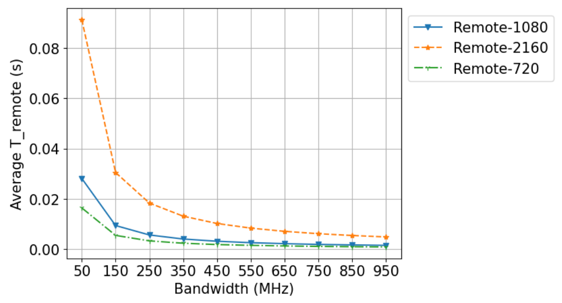
The main trend in Figure 8.1 is that between 50 and 350 MHz, the higher bandwidth significantly improves the QoE-scores of all heuristics that offload remotely. Especially between 150 MHz and 250 MHz, the scores of OB-REMOTE and G-REMOTE jump notably. For bandwidths larger than 350

MHz, only the score of G-REMOTE improves, but less significantly. All other heuristics already reach their maximum before or at 350 MHz, either by having a QoE-score close to the upper bound of 1, or close to 0.3, the upper bound when rendering at most at 1080p ($D_t(1080)/\Theta = 414/1348 \approx 0.3$). The QoE-score of G-REMOTE-ENC drops slightly from 150 MHZ to 50 MHz, but is still outperforming other strategies by more than 0.5. Interestingly, the score of OB-REMOTE is unchanged between 50 and 150 MHz and the scores of OB-REMOTE-ENC and REMOTE-ENC-2160 are almost unaffected overall. To conclude, the increase in bandwidth improves the QoE-score of all rendering strategies, but all strategies eventually reach their performance cap. This means that a limitless bandwidth does not guarantee a QoE-score of 1. Strategies that require less bandwidth (such as all strategies using remote encoding) already achieve this cap early on, at 150 MHz.

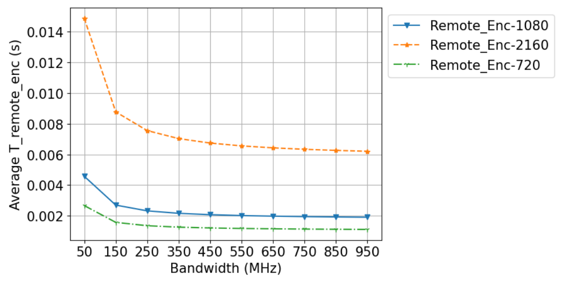
The most striking observation when comparing the QoE-scores of the rendering strategies is the substantial difference in maximally achieved QoE-score. All strategies plateau (or converge to) some performance cap: G-REMOTE, G-REMOTE-ENC, REMOTE-ENC-2160, OB-REMOTE and OB-REMOTE-ENC eventually reach a score close to 1, while RANDOM-1080 and REMOTE-1080 only achieve a score close to 0.3. Local rendering strategies like LOCAL-720 and OB-LOCAL do not exceed a score of 0.18, the maximally achievable score when rendering at most at 720p: $D_t(720)/\Theta = /1348 \approx 0.18$. This implies that rendering strategies that *allow* rendering at 2160p are superior if the circumstances allow for it. The latter condition is important, because the second main observation is that the QoE-scores of OB-REMOTE and REMOTE-1080 plummet below 0 for bandwidths below 150MHz or 50MHz, respectively. Last but not least, the strategies with the best performance make use of remote offloading with encoding. Especially under a bandwidth of 50 MHz, their scores above 0.7 are considerably better than those of other strategies which do not exceed 0.2. Overall, OB-REMOTE-ENC performs the best, closely joined by REMOTE-ENC-2160 and G-REMOTE-ENC.



(a) Average local rendering delay T_{local}.



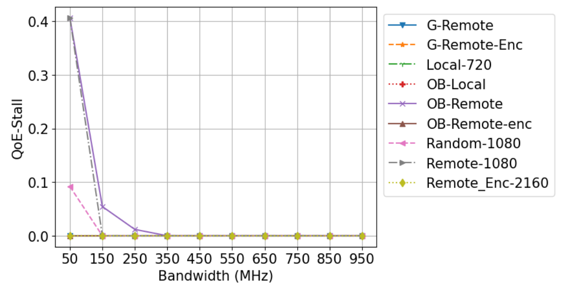
(b) Average remote rendering delay T_{remote}.



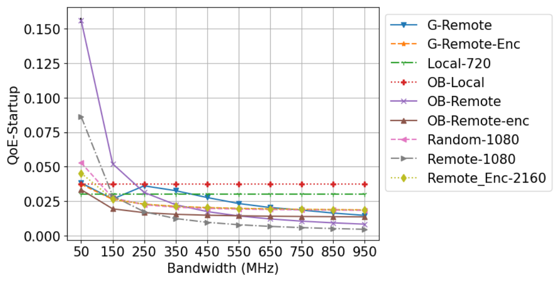
(c) Average remote rendering delay with encoding $T_{remoteenc}$.

Figure 8.2: Average rendering delays over the up- and downlink bandwidths B^u and B^d for constant rendering strategies.

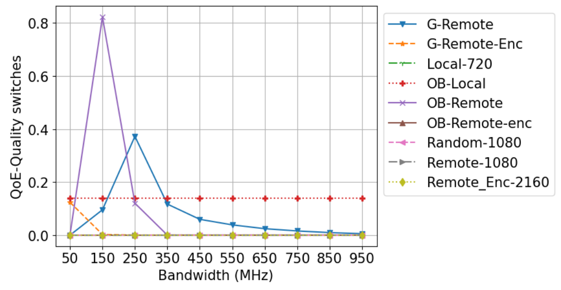
The remainder of the section discusses explanations for the above results. To begin with, the massive impact of the bandwidth between 50 and 250MHz on strategies using remote offloading can be explained by Figure 8.2. The average rendering delays T_{remote} and $T_{remoteenc}$ drop significantly in this range. While $T_{remoteenc}$ is always below $1/\eta = 1/60 \approx 16.7$ ms, T_{remote} is not. At 50 MHz, when rendering at 1080p, the average rendering delay is 17.3 ms. With 10 ms, T_{local} is only below 16.7 ms when rendering at 720p, while the values for 1080p and 2160p are above the $1/\eta$ s. This threshold is important because it marks the moment at which the rendering loop and display loop are equally fast. 60 frames are displayed to the user per second, so if frames are rendered under 16.7 ms, newly rendered frames are entering the buffer quicker than they leave and the buffer can be filled. Consequently, opp-buffer can potentially render frames at higher resolutions than previously possible and, most importantly, the risk of stalls is eliminated. If rendering frames takes longer than 16.7 ms, the system will stall eventually because the rendering process cannot keep up with the playback demand. Figure 8.4a also displays this cross-over point for ob-remote, since the average buffer length on display jumps from near 1 to almost maximum buffer space between 250 and 350 MHz, allowing ob-remote to render at 2160p, hence improving its QoE-quality and consequently its QoE-score.



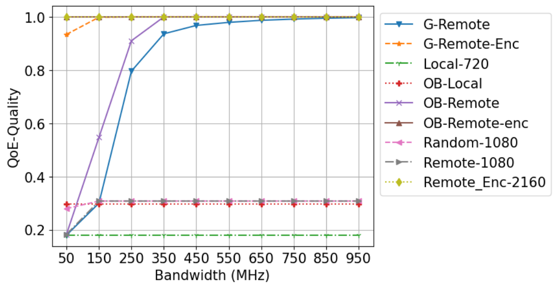
(a) QoE-stall (Percentage of time in stall).



(b) QoE-start-up (start-up delay over 1s).



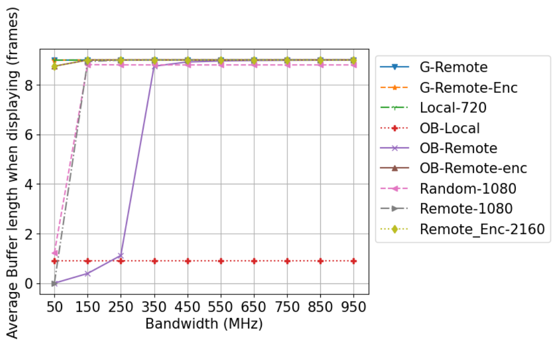
(c) QoE-switches (Percentage of bit-stream behind quality switches).



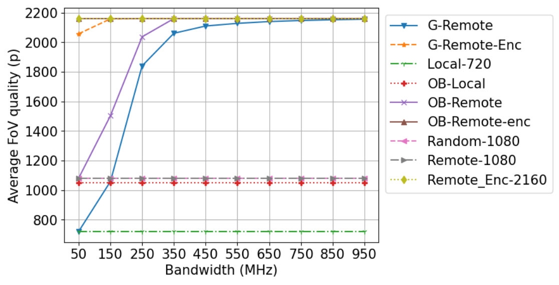
(d) QoE-quality (Percentage of maximal bit-stream achieved).

Figure 8.3: Components of the QoE-score over the up- and downlink bandwidths B^u and B^d.

The breakdown of the QoE-score into its individual component in Figure 8.3 reinforces the above argumentation. The QoE-stall in Figure 8.3a shows that at 50 MHz, the system is in a stall for 40% of time when using remote-1080 or ob-remote. After 350MHz all rendering strategies prevent stalls completely. The explanation for the negative QoE-score of ob-remote can be explained by the high QoE-stall as well as the high QoE-switch scores in Figures 8.3a and 8.3c. Concerning quality switches, we highlight the behaviour of ob-remote and g-remote, whose QoE-switches peak at 150 MHz and 250 MHz, respectively. Figure 8.4b shows that at those bandwidths, the average FoV quality is the closest to the middle between 1080p and 2160p, ergo, the strategy switches most often between 1080p and 2160p. This substantial increase in QoE-switches explains the stagnating QoE-score between 50 and 150 MHz of ob-remote in Figure 8.1, where the increase in QoE-quality (Figure 8.3d) is negated by the increased QoE-switches. Figure 8.3b explains why OB-REMOTE-ENC has a slightly higher QoE-score than REMOTE-ENC-2160, since the former has a lower start-up delay than the latter over all bandwidths. This is explained by the fact that OB-REMOTE-ENC renders the first frames at a lower resolution, since an empty buffer pushes OPP-BUFFER to render at lower resolutions. This does not significantly affect the QoE-quality in Figure 8.3d because the lower quality of the first few frames is eclipsed by the increased quality of all remaining frames. Surprisingly, the RANDOM strategy has the third best QoE-quality score in Figure 8.3d, but the enormous amount of quality switches natural to the RANDOM strategy make it unfeasible (Figure 8.3c).



(a) Average buffer length when displaying a frame.



(b) Average FoV quality chosen.

Figure 8.4: Other performance metrics over the up- and downlink bandwidths B^u and B^d.

The strength of the GREEDY heuristic can be seen in Figures 8.6 - 8.8. Under adverse conditions such as insufficient bandwidth, GREEDY can fall back to its default: local rendering at 720p which always falls under the 16.7 ms threshold. This explains why G-REMOTE outperforms OB-REMOTE significantly at 50MHz in Figure 8.1. As the bandwidth improves, Figures 8.7 and 8.8 show that remote offloading is by far the favoured and at 250MHz even the sole offloading path.

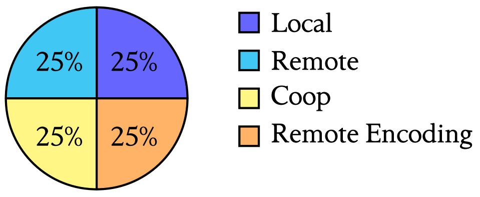


Figure 8.5: Distribution of offloading decisions for RANDOM-1080 $B = 150\text{MHz}$.

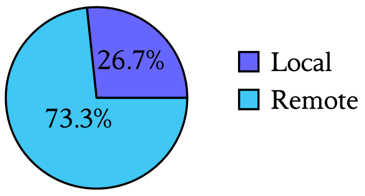


Figure 8.6: Distribution of offloading decisions for G-REMOTE for $B = 50\text{MHz}$.

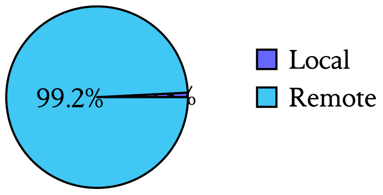


Figure 8.7: Distribution of offloading decisions for G-REMOTE for $B = 150\text{MHz}$.

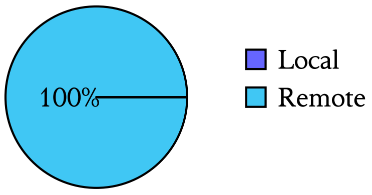


Figure 8.8: Distribution of offloading decisions for G-REMOTE for $B = 250\text{MHz}$.

8.2.2 Impact of maximum buffer capacity

Figure 8.9 presents the main result of this section. It depicts the QoE-score against different values for the maximum buffer size M_B. The number of frames to render in the start-up phase b is set to $\min(3, \max(1, M_B - 1))$ to avoid rendering more frames than fit into the buffer. Similarly to the previous section, we first discuss the impact of M_B and then compare the rendering strategies.

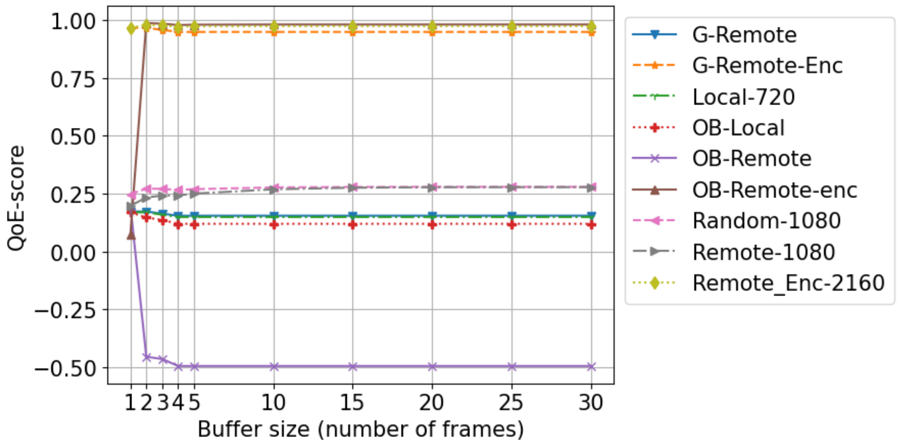


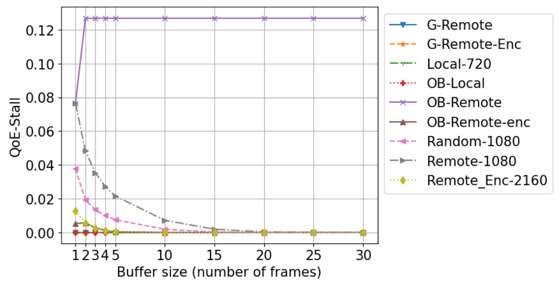
Figure 8.9: QoE-score over the maximum buffer length M_B.

As depicted in Figure 8.9, the maximal buffer size does not have large influence beyond a length of 2 and beyond 5 hardly any change is visible, even for OPP-BUFFER strategies. The only significant impact of the buffer size is between 1 (no buffer) and 2 (1 frame can be stored additionally to the next frame). A buffer size of 2 is significantly improving the QoE-score of OB-REMOTE-ENC and, counter-intuitively, decreasing the performance of OB-REMOTE.

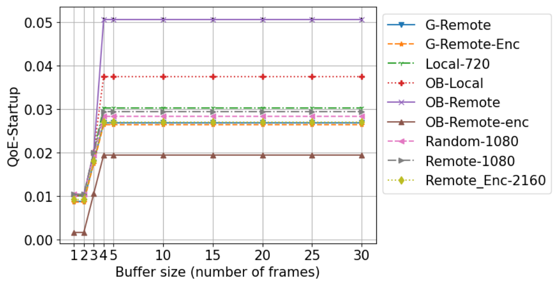
When $M_B \geq 2$, OB-REMOTE-ENC has the highest QoE-score. Surprisingly, the other buffer-related strategies OB-LOCAL and OB-REMOTE are the two worst rendering strategies even with an increased buffer length. OB-REMOTE even has a score below 0. Since we assume a downlink bandwidth of 150 MHz, its poor QoE-score is referable to the explanation in Section 8.2.1. Beyond a buffer size of 2, G-REMOTE-ENC and REMOTE-ENC-2160 are the second and third best performing strategies.

To explain the trends of Figure 8.9, we examine the individual components of the QoE-score in Figure 8.10. The drop of OB-LOCAL, LOCAL-720, G-REMOTE and G-REMOTE-ENC's QoE-scores between a buffer size of 2 and 4 results from the fact that between 2 an 5 the stalling property in Figure 8.10a and the QoE-quality in Figure 8.10d stagnate, but the start-up delay in Figure 8.10b increases since more frames need to be pre-rendered. Moreover, the increase in OP-LOCAL's QoE-quality between a buffer size of 1 and 2 is negated due to an larger rise in the QoE-switches score, leading to an overall decrease in the QoE-score between 1 and 2. The initial increase in QoE-start-up of all strategies in Figure 8.10b is explained by the definition of b, the number of frames to render in the start-up phase. Usually, 3 frames are rendered in the start-up phase and the rendering of the 4th frame commences when the display loop begins. If M_B is less than 4, b is set to $\max(1, B - 1)$ to prevent a buffer overflow.

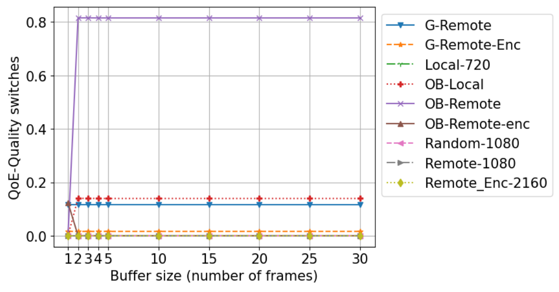
The QoE-quality and QoE-switches scores explain the immense decrease in performance of OB-REMOTE between a buffer size of 1 and above. At $M_B = 1$, the QoE-quality score in Figure 8.10d is around 0.3, equivalent to constantly rendering at a resolution of 1080p. Starting from 2, the QoE-quality increases implying that the strategy renders frames at a higher resolution. However, this is accompanied by an even larger increase of the QoE-switches metric that leads to an overall decrease of the QoE-score. Furthermore, since OB-REMOTE does not fill manage to fill up the buffer (the average buffer length on display is close to 0 in Figure 8.11b), the situation is never improved. This implies that, to achieve a



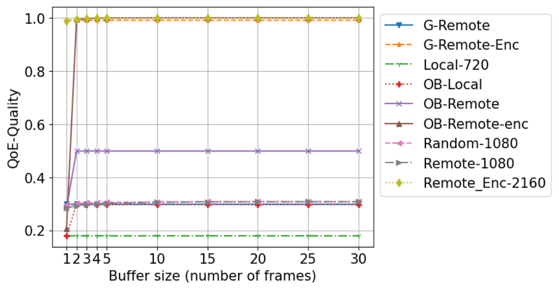
(a) QoE-stall.



(b) QoE-start-up.



(c) QoE-switches.

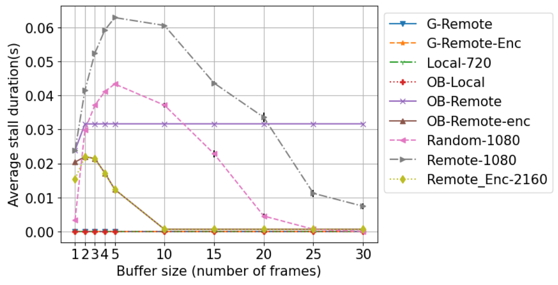


(d) QoE-quality.

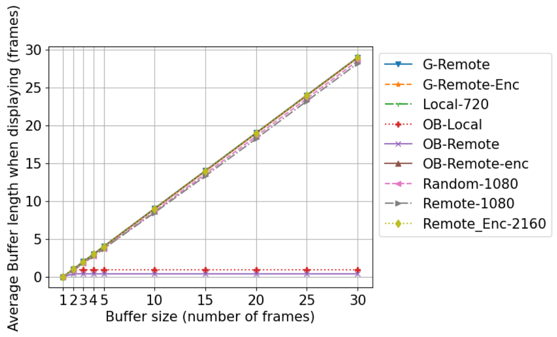
Figure 8.10: Components of the QoE-score over the maximum buffer length M_B.

better QoE-score, strategies should only render at a higher resolution if the system can do so consistently to prevent switching back and forth between two resolutions. For strategies such as remote-enc-2160, ob-remote-enc, remote-1080 and random-1080 that fill the buffer (see Figure 8.11b), the average stall duration in Figure 8.11a first increases and then decreases with additional frames in the buffer, while the QoE-stall score continuously decreases. The increase could signify that stalls become rarer, but last longer, as Figure 8.11a does not consider the total number of stalls but only the average duration of all stalls.

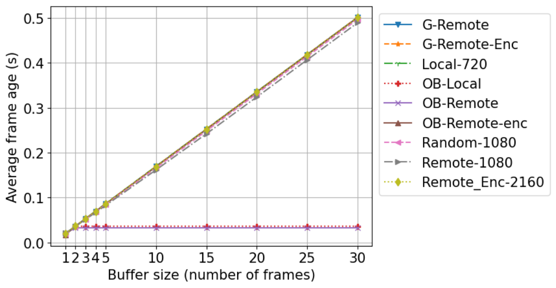
The risk with an increased buffer space is that the age of the individual frames becomes large. Indeed, for all strategies that fill up the buffer space (which have a diagonal line in Figure 8.11b), the average age of a frame increases as well (see Figure 8.11c). Especially for VR gaming where we for now assume that user action is perfectly predicted, this is a significant consideration. Since we can conclude that a buffer size up until 5 is impactful, we can also conclude that investing in prediction algorithms for FoV and user actions is useful. Moreover, we can define the performance goal of such prediction algorithms: with a buffer length of 5, the average frame age is about 0.1 s or 100 ms. According to Guo et al. we must aim for a response delay of at most 25 ms [16], this means the prediction algorithm would need to correctly predict a user's actions up to 75 ms ahead of time.



(a) Average stall duration.



(b) Average buffer length on display.



(c) Average frame age.

Figure 8.11: Other performance metrics over the maximum buffer length M_B.

8.2.3 Impact of the SINR

The results in this section were obtained by drawing the SINR from different uniform distributions. Previous sections assumed SINR values drawn uniformly between 1 and 20 dB. In this section, we draw the SINR values from different uniform intervals: uniformly in $[1,2], [2,3], \ldots, [10,15]$ and $[15,20]$ dB. The x-axis in every figure shows the range chosen. At x, the SNR has been drawn uniformly between the previous value on the x-axis and x. In this way, we keep some uncertainty in the system. This section first presents the impact of the SINR, then compares the rendering strategies before giving potential explanations for the observed results.

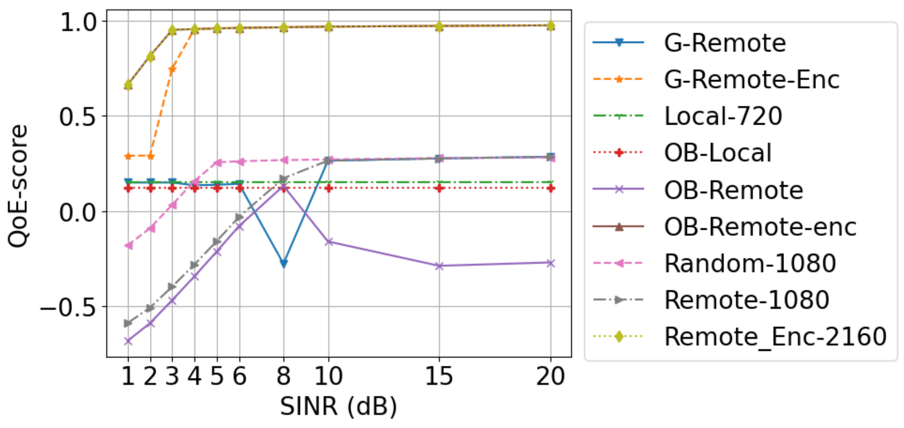
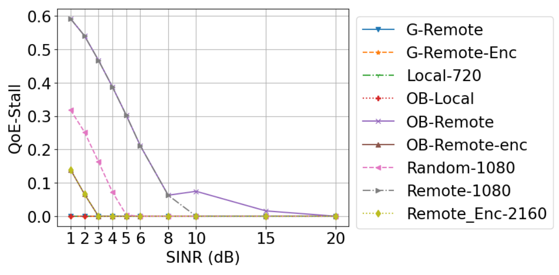


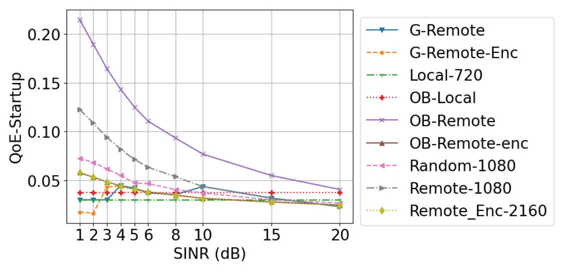
Figure 8.12: QoE score over SINR (in dB).

With better connectivity, i.e larger SINR, the QoE-scores tend to increase. This trend is most significant between 1 dB and 10 dB where the QoE-scores of almost all strategies that offload to the MECs steadily increase. At some point, all strategies have achieved their upper bound and then an increase in γ no longer improves the score. The most surprising observation is that the QoE-score of OB-REMOTE decreases with larger SINR when $\gamma \geq 8$.

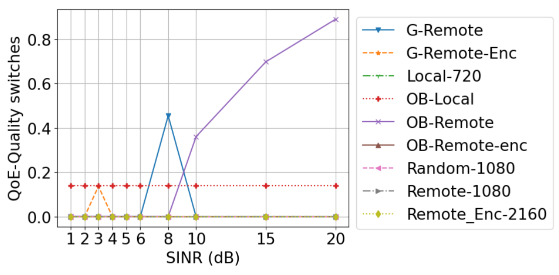
According to the QoE score in Figure 8.12, OB-REMOTE-ENC and REMOTE-ENC-2160 perform the best over all γ. G-REMOTE-ENC is a close second and equals their QoE-score beyond a SINR of 4 dB. Remarkably, the QoE-score stays constant between 1 dB and 2B. Interestingly, RANDOM-1080 is the next best performing strategy if $\gamma \geq 5$ dB. For SINR values below 5 dB the local strategies LOCAL-720 and OB-LOCAL become more competitive, even though their score is still below 0.15. REMOTE-1080 performs as expected between 1 and 10 dB, as its QoE-score continuously rises with a better connectivity. Above 10 dB its QoE-score stagnates. Furthermore, this figure highlights the strength of the GREEDY strategy: compared to the constant baseline REMOTE-1080 and the heuristic OB-REMOTE, G-REMOTE maintains a positive QoE-score even under adverse channel conditions because it can fall back to its default, namely rendering frames locally at 720p. Another interesting observation is that G-REMOTE experiences a sudden decrease of the QoE-score at 8 dB. Finally, the strategy that overall performs poorly is OB-REMOTE which has a negative QoE-score over almost the entire SINR range.



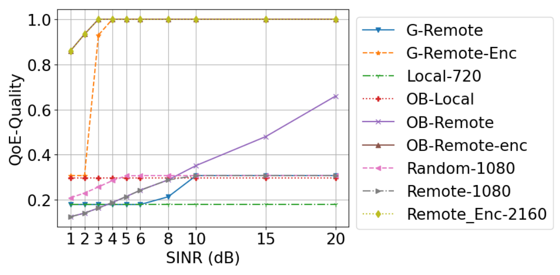
(a) QoE-stall.



(b) QoE-start-up delay.



(c) QoE-switches.



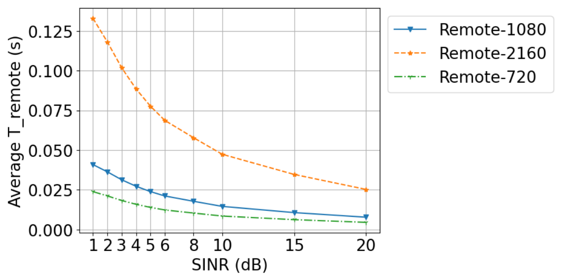
(d) QoE-quality.

Figure 8.13: Components of the QoE-score over SINR (in dB).

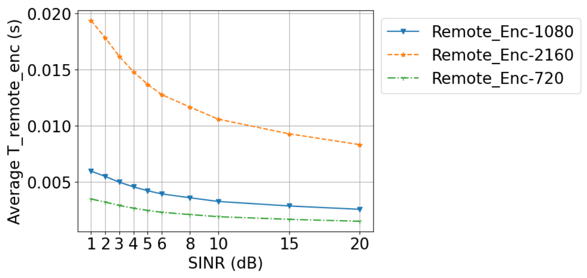
To explain the behaviour of Figure 8.12, we refer to Figure 8.13 highlighting the evolution of the individual QoE components under different SINR values. First of all, the upper bound on the different QoE-scores can be explained by Figure 8.13d: most strategies converge towards always rendering at a fixed resolution. Rendering at 2160p equals a QoE-quality of 1, while 1080p implies 0.3 and 720p 0.18. We assume that OB-REMOTE trends to the QoE-quality of 1, but it does not converge yet in the simulated SINR range.

The dip of G-REMOTE's QoE-score results from the peak in the QoE-switches score in Figure 8.13c. At 8 dB, the QoE-quality in Figure 8.13d increases, implying that the strategy renders at higher quality. The corresponding increase in QoE-switches shows that the higher resolution cannot be supported consistently and even causes a deteriorating QoE-score. Furthermore, we can see an equivalent peak of QoE-switches in the other GREEDY strategy at 3 dB. However, G-REMOTE-ENC's QoE-quality rises enormously at the same time, leading to an overall improvement of the QoE-score. The low QoE-score of local strategies LOCAL-720 and OB-LOCAL primarily results from the low QoE-quality in Figure 8.13d. Even though OB-LOCAL has a higher QoE-quality, the overall QoE-score is lower compared to LOCAL-720 because of the many quality switches. This is yet another example why switching to higher qualities should only be done if it can be pulled off consistently, otherwise the overall performance is deteriorated due to the many switches.

The average T_{remote} and $T_{remoteenc}$ of the constant base line strategies are shown in Figures 8.14a and 8.14b. The rendering delay $T_{remoteenc}$ is below the threshold of 16.7ms for all resolutions lower or equal to 1080p for all γ. When rendering at 2160p it is below the 1/60 threshold if $\gamma \geq 5$ dB. For the delay T_{remote} this only happens for 720p when $\gamma \geq 5$ dB and when $\gamma \geq 10$ dB for 1080p and never when rendering at 2160p. This behaviour explains the bad performance of OB-REMOTE which after 8 dB, counter-intuitively, performs worse with a better SINR. At 10 dB, the rendering process (at 1080p) is quicker than the framerate, such that the buffer can fill up. Consequently, rendering a frame at 2160p becomes feasible if the buffer is full enough. So while the QoE-quality score increases with γ in Figure 8.13d as expected, the QoE-switches score increases even harder in Figure 8.13c, resulting in an overall decrease of the QoE-score. At some point, the QoE-switches score is above 90% meaning that the heuristic switches quality resolution for almost every frame, which cannot be a comfortable experience for the user.



(a) Average remote rendering delay T_{remote}.



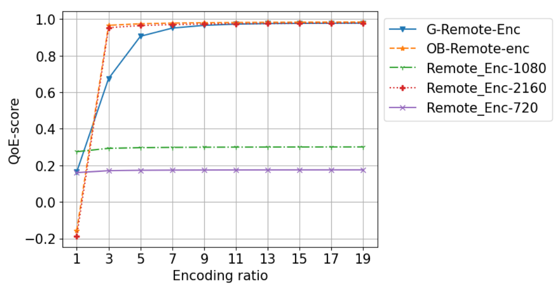
(b) Average remote rendering with encoding delay.

Figure 8.14: Average rendering delays over SINRs at $B^u = 150$ MHz for constant rendering strategies.

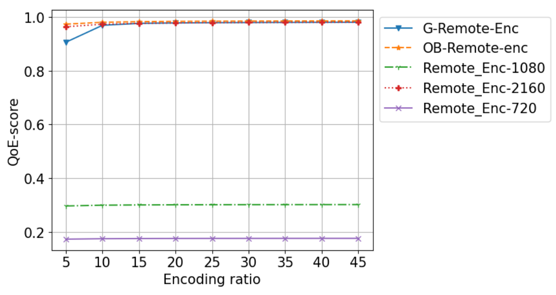
8.2.4 Impact of the remote encoding ratio

This section studies the impact of the remote encoding ratio k on all strategies that use remote offloading with encoding. These strategies showed the best performance in previous sections, so we want to examine them in more detail. First, this chapter sheds light on the impact of the encoding ratio k. To recapitulate, an encoding ratio of k means that the fully rendered frame is compressed to $1/k$th's of its original size before transmitting the file from the MECs to the HMD. Furthermore, compared to T_{remote}, $T_{remoteenc}$ has an additional encoding delay of $\frac{D}{z_c \cdot b_{z_c}}$ and an additional decoding delay of $\frac{1}{k}\frac{D}{z_l \cdot b_{z_l}}$. Secondly, this section discusses an important assumption for the calculation of $T_{remoteenc}$. We show that the QoE-score significantly drops if we make a different assumption on how to calculate the decoding delay at the HMD.

From Figure 8.15, we can see that the best QoE-score for all heuristics is already achieved at an encoding ratio of 11. For all heuristics except G-REMOTE-ENC, the strategies already reach their respective maximum when $k \geq 3$. At $k = 1$, so without compression, the QoE-scores plummet significantly as expected from the results of strategies like REMOTE-1080 or G-REMOTE in previous sections. Especially OB-REMOTE and REMOTE-ENC-2160, which render primarily at 2160p, have devastating QoE-scores below 0. Finally, it is noteworthy that REMOTE-ENC-1080 actually outperforms G-REMOTE-ENC at $k = 1$.

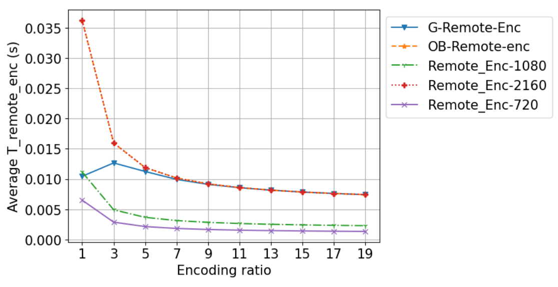


(a) Smaller encoding ratios.

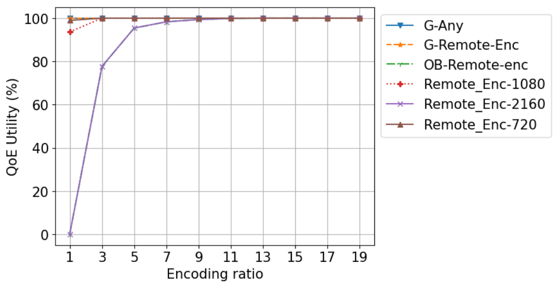


(b) Larger encoding ratios.

Figure 8.15: QoE-score over smaller and larger encoding ratios k.



(a) Average $T_{remoteenc}$.



(b) QoE utility.

Figure 8.16: Other performance metrics over encoding ratios k.

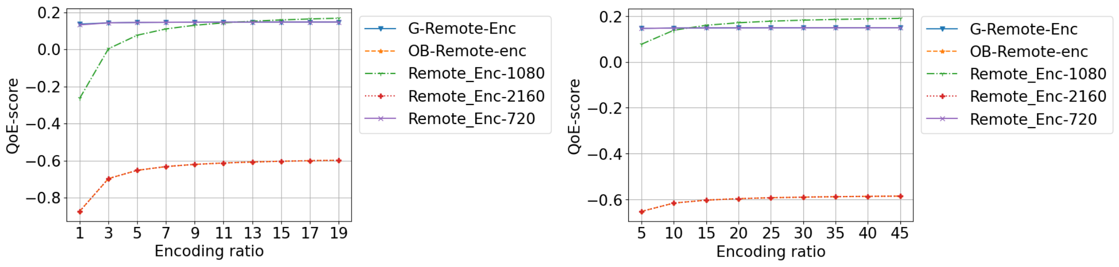
Figures 8.16a shows indeed that, when $k \geq 3$, the average rendering delay of all strategies is below 16.7 ms, explaining why the QoE-score does not change significantly beyond this point. However, we can observe that although the average is below 16.7ms, the QoE utility in Figures 8.16b is only around 100% for all strategies when $k \geq 11$. To recall, the QoE utility stands for the percentage of frames whose rendering delay is below 16.7ms. These two observations imply that REMOTE-ENC-2160's rendering delay highly depends on the SINR value γ when $k < 11$, because on average the rendering delay meets the 1/60 s threshold, but there must be a high variance due to the randomness in γ.

An important discussion point when considering heuristics that use encoding when remotely offloading, is that we *assume* that the HMD does not need to process the full file size when decoding the frame but only the compressed file size. In Chapter 3, we noted that decoding is one of the main limiting factors of the rendering delays. In Guo et al.'s model [16], the MECs always need to process 10MB to compress the frame, the compressed frame always has a size of 2 MB (so only $k = 5$) and the HMD needs to process $(5/9) \cdot q_i^2$ bytes to decode the frame. In other words, in Guo et al.'s model, the decoding delay is not dependent on the compression ratio k.

To see the influence on the QoE-score when we remove the compression coefficient k from the decoding delay computations, we simulated the same setting as above using the following $T^*_{remoteenc}$:

$$T^*_{remoteenc} = \underbrace{\frac{D_u}{R^u}}_{\text{Uplink transmission}} + \underbrace{\frac{C}{z_c}+}_{\text{render fore- and background}} + \underbrace{\frac{D}{z_c \cdot b_{z_c}}}_{\text{Compression latency}} + \underbrace{\frac{D_k}{R^d}}_{\text{Downlink Transmission}} + \underbrace{\frac{D}{z_l \cdot b_{z_l}}}_{\text{Decoding compressed frame}}, \quad (8.1)$$

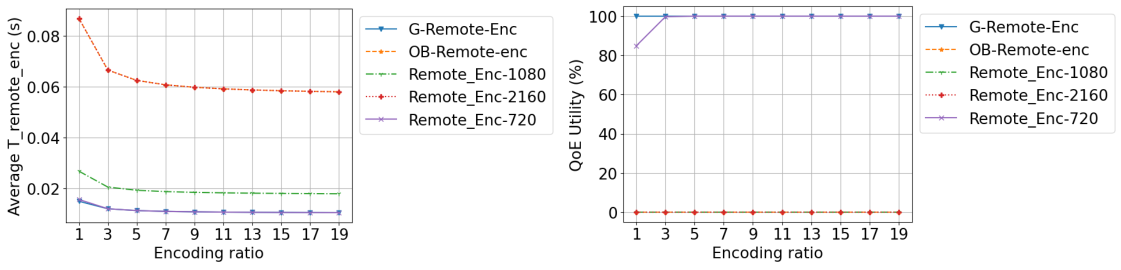
where R^u and R^d are the up- and downlink transmission rates of the user, D and C the data size and computational requirement of the frame, D_u the data size needed for the uplink communication about the foreground rendering, $D_k = \frac{D}{k}$ the file size of the compressed frame depending on the compression ratio k and z_c the processing power at the MECs allocated to the user. The difference to $T_{remoteenc}$ in Equation (4.8) is only in the last term.



(a) Smaller encoding ratios (b) Larger encoding ratios

Figure 8.17: QoE-score over encoding ratios k using $T^*_{remoteenc}$.

Figure 8.17 shows the QoE-scores for the simulations with $T^*_{remoteenc}$. These plots show that even for an encoding ratio of 45, the QoE-score does not exceed 0.2. Interestingly, we can again note that the encoding ratio does not influence the QoE-score significantly anymore when $k \geq 11$. Compared to the scores around 1.00 in Figure 8.15, a score of 0.2 is a strong deterioration.



(a) Average $T^*_{remoteenc}$. (b) QoE utility.

Figure 8.18: Other performance metrics over encoding ratios k.

The explanation for the deterioration can be found in Figure 8.18a showing that $T^*_{remoteenc}$ never falls below the 16.7ms threshold for resolutions higher or equal to 1080p. Only strategies that settle for a resolution of 720p, such as REMOTE-ENC-720 and G-REMOTE-ENC, have QoE-utilities of 100% in Figure 8.18b. This implies that they never cause stalls which explains why these two strategies have the highest QoE-score despite the lower resolution of 720p.

The true decoding delay likely lies in between these two extremes. In $T^*_{remoteenc}$, the HMD needs to again process the entire file size D to decode it. However, in our model this is equivalent to *rendering the frame from scratch* with a computational requirement of C, because the values C and D are linked in such a way that the processing delays are equal: $D/(b_{zl} \cdot z_l) = C/z_l$. Consequently, when rendering a frame and decoding a frame take equally long, it logically never makes sense to offload remotely with encoding because it only introduces additional delays such as the compression delay or the transmission delays compared to rendering locally. To conclude, strategies using encoding have the biggest potential among all offloading strategies, if the decoding delay decreases proportionally to the encoding ratio k.

8.3 Do dynamic rendering strategies outperform static rendering strategies in terms of QoE-score?

Based on the results in Section 8.2, we can draw the following conclusions regarding RQ 1:

1. Are dynamic rendering strategies outperforming static rendering strategies in terms of QoE-score?

   In all scenarios, static rendering strategies have either been outperformed or matched by a OPP-BUFFER or GREEDY strategy. However, we cannot conclude that the evaluated dynamic heuristics always have a significantly higher QoE-score than the static strategies. In fact, REMOTE-ENC-2160, which always renders remotely at 2160p with encoding has a QoE-score close to 1.00 in most scenarios, which exceeds the QoE-score of G-REMOTE, OB-REMOTE and OB-LOCAL at 150MHz in Figure 8.1 by more than 0.5. This result shows that the influence of the chosen offloading paths is larger than the dynamic decision-making itself. The main advantage of dynamic strategies is that they can adjust to rapid system deterioration, such as poor SINR in Figure 8.12, where GREEDY could decide to render locally at a lower resolution instead of risking a performance deterioration by remotely offloading such as REMOTE-1080.

(a) & (b) Do strategies that consider current channel conditions and strategies that consider the buffer space achieve a high QoE-score?

Both types of heuristics have performed reasonably well. We cannot conclude that one type of heuristic is significantly better than the other because it depends on the paths they favour and on the system parameters. However, we notice that whenever the user has a decent data rate available (high bandwidths or good SINR), OB-REMOTE-ENC is (one of) the best performing strategy. However, OPP-BUFFER can also suffer from many quality switches resulting in a devastating performance, as we could see from OB-REMOTE. The main advantage of GREEDY is that it can fall back to local rendering under adverse channel conditions (low SINR). However, it generally performs worse than its opp-buffer counterpart. Only G-REMOTE outperforms OB-REMOTE when the bandwidth is below 250MHz.

(c) What offloading path (local, remote, coop and remote rendering) lead to high QoE-scores?

Remote rendering with encoding and decoding outperforms other offloading paths significantly in our current model. However, even in Figure 8.17a, where the decoding delay is proportional to the full frame size, the QoE-score is around 0.2. While it is a massive deterioration from the previous score of almost 1, it is similar to the other heuristics at 150MHz in Figure 8.1 and still better than rendering locally (LOCAL-720). This observation leads us to conclude that a simple rendering heuristic such as always remotely offloading with encoding (REMOTE-ENC-2160) is sufficient as a rendering strategy. Investing in lightweight compression algorithms that can be quickly decoded is more beneficial than complex offloading algorithms to achieve better performance.

Chapter 9

Discussion and Future Directions

This chapter first reflects on the system model and the performance evaluation before providing inspiration for future rendering heuristics based on the gained insights.

First of all, while the results for the remote encoding paths are the most promising, we have to take the results with a grain of salt because the encoding and decoding delays are the most difficult to model correctly. While we have some empirical evidence that the relations could be modelled in this way (see Section 4.4.4), the numbers measured by [18] are significantly larger than our values. This discrepancy could result from the fact that they performed their experiments on a consumer desktop and phone. In contrast, we assume the processing capabilities of the MECs and the HMD to be larger by taking the values proposed by Guo et al. [16]. In order to verify the model and determine appropriate encoding and decoding delays, future work could consist of replicating Lai et al.'s measurement experiments with our system set-up. To avoid the challenges when testing over the actual cellular network, the measurements can be reduced to only measuring the rendering, encoding and decoding delays instead of replicating the entire offloading system. The results in Section 8.2.4 show that reducing the decoding delay can significantly improve the QoE-scores of strategies that remotely offload with encoding. Hence, the modelling assumptions regarding the decoding delay deserve serious attention in future work.

Another point of discussion is the distributed nature of the offloading decision. All our proposed heuristics can be implemented in a distributed fashion on every end device and do not require a central decision maker. Distributivity was relevant because a centralised decision (at, for example, the edge server) will encounter difficulties in a practical set-up. Not only would devices have to interoperate (all VR brands would have to stick to the same offloading decision-making procedure that the edge understands) but also edge operators and application-makers would have to cooperate and share information. However, despite the distributed nature, some challenges concerning information accessibility must be overcome before applying the proposed heuristics in practice. For example, GREEDY requires estimates of the channel quality and the available bandwidth and processing power at the MECs to estimate the rendering delay. Communicating this cross-layer information is a challenge in itself. Moreover, we must tackle the question, "Which entity of the HMD is responsible for the offloading decision?" Here, several answers are possible since one option could be the game developers who know the game best, thus, know better how to divide a frame into fore- and background or which actions the player is likely to do next. Another option could be the HMD's manufacturer, which has a better knowledge of the device's computing and networking capabilities. A third option could be an external party joining the information of both. All in all, we acknowledge that the proposed heuristics serve as proof of concept, and many practical challenges would need to be resolved if it was to be applied in practice.

One of the main limitations of our model is that we *assume* the user's action to be perfectly predictable both concerning the FoV and the user input. In a VR game, the latter influences the game-play content, so what needs to be rendered. If we aim to show the effect of user actions within a response time of 20ms following the framerate of 60 FPS [16], then there can only be a single frame shown between the action input and the display of the new frame showing the corresponding reaction on screen. This scenario is equivalent to a buffer size of 2. Figure 8.9 showed that maximum buffer sizes beyond 2 have only a minor influence on the QoE-score of strategies, such that the impact of this assumption is minor. Secondly, we assume perfect FoV prediction with a predefined number of tiles lying in the FoV. However, bitrate adaption schemes exist that use other tiling schemes and ways to predict the FoV and to decide at which resolution to fetch the individual tiles [17]. Moreover, FoV prediction schemes are *predictions*, so they might not be 100% accurate. To mitigate this assumption, future studies could investigate the impact of tiling schemes and include an error term whenever the FoV is wrongly predicted. A second main limitation is that our results are based on simulations because it is unfeasible to determine the optimal strategy for the optimisation problem. This is reinforced by the fact that system parameters significantly impact the system, so even an optimal strategy under some system parameters (available bandwidth, processing power, etc.) might not be optimal anymore under others. Moreover, we assume that such system parameters are immutable and correctly communicated to rendering strategies. Future work could investigate the impact of delayed or erroneous information about system parameters such as the available processing power at the MECs, the available bandwidth or the SINR ratio. The perturbation coefficient σ that can alter the SINR value used in the estimates of GREEDY strategies is a starting point to evaluate the influence of inaccurate system information.

Furthermore, since this work is based on simulations which are in turn based on a model of the real world, a natural limitation of the research is that, intrinsically, the results show the performance evaluation of the model. These results might not directly apply to reality, and we must critically assess our system model's shortcomings and abstractions to determine this. The main simplification in our model is to draw independent SINR values from a uniform distribution, whereas, in practice, subsequent SINR values are probably correlated. Models with more realistic SINR distributions exist that calculate SINRs from the HMD's transmission power, its distance to the base station and ambient noise and interference models [16]. In such a model, it would be possible to integrate other performance metrics, such as the energy consumption of the HMD, which strongly depends on the HMD's transmission power. Furthermore, the best method to verify the system model would be to conduct real-life experiments and compare whether the empirical results match the simulation results. However, real-life experiments can be costly, time-intensive and risk adding implicit dependencies on specific materials and applications, making it harder to generalise the findings and compare them to a 'sterile' simulation. Instead of implementing the entire system, individual aspects of the model could be replaced with empiric measurements. For example, SINR values are drawn from a random distribution in the current performance evaluation. However, they could be taken from actual SINR traces as done by Mehrabi et al. [28]. Additionally, instead of calculating all data volumes and intermediate delays as in Sections 4.3 and 4.4, the simulator could use empirically measured rendering, encoding and decoding delays, as well as application bitrates. A specific HMD and VR application would need to be chosen to measure these different aspects.

Finally, in the current setting, some delays are assumed to be negligible similar to simulations in related works. For a practical implementation, future work would have to determine whether this assumption is indeed valid. Some actions for which we assume negligible delays are: adding a frame to the buffer, removing a frame to the buffer and displaying it on the HMD, determining the next frame that needs to be rendered, queuing delays before transmissions or making the rendering decision. However, we are optimistic that the delays are indeed minor because many consist only of a few operations or instructions. Future work could empirically investigate how long it takes to make a decision. In our case, we can expect the computational complexity of the static and dynamic rendering strategies and the delay accordingly to be reasonably small. However, when employing a more complex offloading strategy based on deep reinforcement learning and game theory, such as Guo et al. [16], the complexity is higher, and decision-making might not be instant.

We wrap up this chapter by presenting other directions for designing rendering heuristics that could be investigated in the future.

Future direction: Other rendering heuristics

One of the main results in Chapter 8 is that dynamic strategies risk a significant decrease of the QoE-score under circumstances where rendering at higher resolutions is sometimes possible but not always. Then, the increase in QoE-quality is overshadowed by the decrease in QoE-switches. Hence, a new type of rendering strategy can minimise this problem by only rendering at higher resolutions if the system allows it to do so consistently. For example, strategies could only adapt the quality of every xth frame instead of every single frame. Alternatively, OPP-BUFFER could check if the currently buffered frames would allow rendering at least x frames at a higher resolution by calculating whether $1/60 \cdot \textbf{length}(\text{buffer}) + x - 1) > x \cdot T_q$, where T_q is the expected rendering delay at a higher resolution q, to prevent switching to a higher resolution for only a single frame. Furthermore, heuristics that account for the history until the decision moment could be worthwhile in settings where the SINRs are time-dependent and correlated.

What is more, so far, the proposed heuristics have been designed to maximise the QoE-score by reducing stalling times and increasing the QoE-quality. However, they do not *directly* optimise based on the QoE-score. Future work could look into rendering strategies whose goal is the more direct maximisation of the QoE-score at every single frame. Here below, we provide a possible heuristic based on the idea that Equation (4.19) can be rewritten to the following, where the influence of every frame i towards the final QoE-score becomes more apparent:

$$\underbrace{\frac{D^1}{\Theta} - T_o^1}_{\text{First frame}} + \underbrace{\sum_{i=2}^{i^*} \left(\frac{D^i}{\Theta} - \frac{|D^i - D^{i-1}|}{D} - T_o^i \right)}_{\text{During start-up phase}} + \underbrace{\sum_{i=i^*+1}^{N_T} \left(\frac{D^i}{\Theta} - \frac{|D^i - D^{i-1}|}{D} \right)}_{\text{Others}} - \underbrace{\frac{T_S}{T}}_{\text{QoE-stall}},$$

where Θ is the maximally achievable bitrate, D is the total number of bits played, T is the total play time, T_S the total stall duration, i^* is the ID of the last frame of the start-up phase, N_T is the last displayed frame and T_o^i the rendering delay of frame i using the offloading path o. Only the rendering delays of the terms in the start-up phase contribute to the QoE-start-up. Moreover, the first term cannot incur a quality switch. The only QoE-metric that depends on the rendering of previous frames is the QoE-stall metric. To minimise the stall duration at the frame level, the system state needs to monitor whether a stall occurs before the new frame's rendering is finished. If a stall happens, the rendering decision can minimise the total stall duration by minimising the expected rendering delay of the new frame.

Then, for example, a heuristic that more directly maximises the QoE-score chooses the offloading mode and resolution quality to maximise:

$$\underbrace{\kappa_1' \cdot D^i}_{\text{Quality of the frame}} - \underbrace{\kappa_2' \cdot |D^i - D^{i-1}|}_{\substack{\text{Quality difference}\\ \text{to previous frame}}} - \underbrace{\begin{cases} \kappa_3' \cdot T_o^i, & \text{during start-up phase} \\ 0, & \text{else} \end{cases}}_{\text{Rendering delay during start-up}} - \underbrace{\begin{cases} \kappa_4' \cdot T_0^i, & \text{during a stall} \\ 0, & \text{else} \end{cases}}_{\text{Rendering delay when stalling}},$$

with D^i the size and T_o^i the rendering delay of frame i. For the above, the system state needs to include a variable that indicates whether the system is currently in a stall or not. The coefficients κ_1', κ_2', κ_3' and κ_4' can be used to balance the magnitudes of the individual components and adjust their significance.

We emphasise that the above heuristic would not guarantee an optimal QoE-score. Firstly, the sequential decision process cannot reach an optimum that requires earlier frames to adopt sub-par rendering decisions for a later benefit. Secondly, even if a *Nash Equilibrium* is reached, it might not be the optimum (a typical game theory example showing this principle is the *Prisoner's Dilemma*) [33]. A rendering strategy (the collection of every frame's rendering decision) results in a Nash Equilibrium if the QoE-score cannot improve by changing one frame's rendering decision if all other decisions stay the same.

To conclude, there are still many rendering heuristics that can serve as inspiration for future research directions.

Chapter 10

Conclusion

VR games have stringent computation and latency requirements that the HMD cannot handle by itself. Offloading to MEC servers at the base station can alleviate this computational burden while assuring low delays due to the proximity of the servers to the HMD and due to 5G and beyond data rates. Rendering strategies determine at which resolution quality to render new frames and where to do the computations (locally or at the MECs). A user's quality of experience increases with higher resolution qualities since the VR immersion becomes more realistic, and the risk of cybersickness is reduced. However, higher resolutions require more computations, hence the rendering takes longer, and the HMD risks not keeping up with the framerate required for a smooth viewing. We measure the overall quality of experience by the QoE-score, which factors in the quality of frames, the experienced stalls, the quality differences between frames and the start-up delay.

While many algorithms come up with strategies to stream VR videos or render VR games in real-time, the complexity of the algorithms and the many factors involved make it difficult to explain the performance of a strategy and to determine potential bottlenecks. Since the networking landscape is constantly changing and processing capabilities increase, it is valuable to know which part of the system limits further performance enhancements. Hence, we tackle the problem by focusing on the explainability of the results. Moreover, to our knowledge, no existing work on rendering strategies for VR games considers the video streaming-like nature of VR games. Therein lies the main contribution of this thesis: we consider the VR gaming process holistically by studying the effects of rendering strategies not on a frame-by-frame basis but by simulating an HMD whose rendering process needs to keep up with the steady demand for rendered frames and where stalls and quality switches deteriorate the QoE.

To answer the leading research questions "Do dynamic rendering strategies outperform static rendering strategies in terms of QoE-score?" (RQ 1) and "What is the impact of system parameters?"(RQ 2), two dynamic heuristics, GREEDY and OPP-BUFFER, have been compared to static rendering strategies and evaluated under different system parameters using a simulation framework that takes into account the 'streaming'-aspect of a VR game. The main results and conclusions are as follows:

1. Rendering frames under 1/60s eliminates the risk of stalls when displaying the game at 60 FPS. If, in addition, the frame can be rendered consistently at 2160p, the QoE-score is close to 1.

2. The proposed dynamic heuristics do not outperform all static rendering strategies, because REMOTE-ENC-2160 outperforms all GREEDY and OPP-BUFFER strategies that do not consider remote offloading with encoding. Nonetheless, dynamic strategies have displayed advantages over static strategies. For example, considering the channel quality was beneficial whenever the network quality did *not* allow for offloading. Under such circumstances, GREEDY defaulted to rendering frames locally, keeping its QoE-score from deteriorating drastically under poor SINR or bandwidth conditions.

3. The factor with the most impact on the rendering strategies is the favoured offloading path. All rendering strategies that concentrate on offloading remotely with encoding, even the static REMOTE-ENC-2160, outperform all other strategies. However, this result is linked to our main discussion point, namely how to realistically model the decoding delay because the result only holds when the encoding delay is proportional to the data size of the compressed frame. If it is proportional to the raw frame size, the QoE-scores of all strategies using remote rendering with encoding deteriorate significantly.

4. The evaluated rendering strategies have an intrinsic upper bound on their QoE-score that highly depends on the resolution quality of frames. Despite better system conditions (such as bandwidth or SINR), the QoE-score stagnates if a strategy does not consider rendering at higher resolutions or if such a rendering delay would exceed the 1/60 s threshold under the given system conditions. Hence, blindly increasing the available data rates does not necessarily imply an improved QoE-score.

5. A large number of quality switches between frames deteriorates the QoE-score significantly. Both the GREEDY and OPP-BUFFER strategies can suffer from this problem when the buffer or estimated channel conditions *sometimes* allow rendering at higher resolutions but cannot keep it up consistently. Then, the strategy risks switching back and forth between two resolutions, and the substantial QoE deterioration due to the many quality switches negates the benefit of the higher quality. Hence, strategies should only render at higher quality resolutions if it can be done consistently.

The last point is an opportunity for future work to investigate other rendering heuristics that can mitigate the many quality switches. Other future directions include an elaborate verification of model assumptions and the inclusion of more complex models for SINR values or empirical measurements thereof.

To conclude, edge networks can provide promising opportunities for VR gaming applications by taking over parts of the rendering process. The MEC server's additional processing power and data rates of 5G and beyond networks can accommodate the rendering of frames at 4K resolutions or 2160p while satisfying a framerate of 60 FPS if the HMD can decode compressed frames quick enough.

Bibliography

[1] Mahbuba Afrin, Jiong Jin, Akhlaqur Rahman, Ashfaqur Rahman, Jiafu Wan, and Ekram Hossain. Resource Allocation and Service Provisioning in Multi-Agent Cloud Robotics: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials*, 23(2):842–870, apr 2021.

[2] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 14(1):1–16, jan 2018.

[3] Eva Cerezo, Frederic Pérez, Xavier Pueyo, Francisco J Seron, and François X Sillion. A survey on participating media rendering techniques. *The Visual Computer*, 21(5):303–328, 2005.

[4] Qi Cheng, Hangguan Shan, Weihua Zhuang, Lu Yu, Zhaoyang Zhang, and Tony QS Quek. Design and analysis of mec-and proactive caching-based 360 mobile vr video streaming. *IEEE Transactions on Multimedia*, 2021.

[5] Cineversity. Vr resolutions output specifications. YouTube at https://www.cineversity.com/wiki/VR_Resolutions_%26_Output_Specifications/. Accessed: 2022-08-26.

[6] European Commission. Regulation (EU) 2016/679 (General Data Protection Regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[7] Roberto Irajá Tavares Da Costa Filho, Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Filip De Turck, Luciano Paschoal Gaspary, Maria Torres Vega, and Marcelo Caggiani Luizelli. Predicting the performance of virtual reality video streaming in mobile networks. *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys 2018*, 18:270–283, jun 2018.

[8] Dbpixelhouse. Oculus rift. https://www.dbpixelhouse.com/products/oculus-rift. Accessed: 2022-08-19.

[9] T.R. De Oliveira, M.M. Da Silva, R.A.N. Spinasse, G.G. Ludke, M.R.S. Gaudio, G.I.R. Gomes, L.G. Cotini, D. Vargens, M.Q. Schimidt, R.V. Andreao, R.V. Andreao, and M. Mestria. Systematic Review of Virtual Reality Solutions Employing Artificial Intelligence Methods. In *ACM International Conference Proceeding Series*, pages 42–55, 2021.

[10] Digi. Understanding lte signal strength values. https://www.digi.com/support/knowledge-base/understanding-lte-signal-strength-values. Accessed: 2022-08-23.

[11] Rudzidatul Akmam Dziyauddin, Dusit Niyato, Nguyen Cong Luong, Ahmad Ariff Aizuddin Mohd Atan, Mohd Azri Mohd Izhar, Marwan Hadri Azmi, and Salwani Mohd Daud. Computation offloading and content caching and delivery in vehicular edge network: A survey. *Computer Networks*, 197:108228, 2021.

[12] Elsevier. Scopus. https://www.scopus.com/, 2004. Accessed: 2022-02-28.

[13] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A. Rahmanian. Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing 2019 18:1*, 18(1):1–42, sep 2019.

[14] Rajkumar Gunasekar and Naveen Kumar Chandramohan. Development of image acquisition system to eleminate blind spot of a-pillar. *International Journal of Scientific Research & Management Studies*, 8:145–151, 10 2018.

[15] F. Guo, F.R. Yu, H. Zhang, X. Li, H. Ji, and V.C.M. Leung. Enabling Massive IoT Toward 6G: A Comprehensive Survey. *IEEE Internet of Things Journal*, 8(15):11891–11915, 2021.

[16] Fengxian Guo, F. Richard Yu, Heli Zhang, Hong Ji, Victor C.M. Leung, and Xi Li. An Adaptive Wireless Virtual Reality Framework in Future Wireless Networks: A Distributed Learning Approach. *IEEE Transactions on Vehicular Technology*, 69(8):8514–8528, aug 2020.

[17] Jeroen Van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Tile-based adaptive streaming for virtual reality video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(4):1–24, 2019.

[18] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 409–421, New York, NY, USA, 2017. Association for Computing Machinery.

[19] Mohammed Laroui, Boubakr Nour, Hassine Moungla, Moussa A. Cherif, Hossam Afifi, and Mohsen Guizani. Edge and fog computing for IoT: A survey on current research activities & future directions. *Computer Communications*, 180:210–231, dec 2021.

[20] Bin Li, Zesong Fei, Jian Shen, Xiao Jiang, and Xiaoxiong Zhong. Dynamic offloading for energy harvesting mobile edge computing: Architecture, case studies, and future directions. *IEEE Access*, 7:79877–79886, 2019.

[21] Q. Li, D. Wang, and H. Lu. A Cooperative Caching and Computing-Offloading Method for 3C Trade-Off in VR Video Services. *IEEE Access*, 9:124010–124022, 2021.

[22] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '01, page 238–246, New York, NY, USA, 2001. Association for Computing Machinery.

[23] Hai Lin, Sherali Zeadally, Zhihong Chen, Houda Labiod, and Lusheng Wang. A survey on computation offloading modeling for edge computing. *Journal of Network and Computer Applications*, 169:102781, nov 2020.

[24] Xiaoling Ling, Jie Gong, Rui Li, Shuai Yu, Qian Ma, and Xu Chen. Dynamic Age Minimization with Real-Time Information Preprocessing for Edge-Assisted IoT Devices with Energy Harvesting. *IEEE Transactions on Network Science and Engineering*, 8(3):2288–2300, jul 2021.

[25] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys and Tutorials*, 21(4):3133–3174, oct 2019.

[26] Wenliang Mao, Zhiwei Zhao, Zheng Chang, Geyong Min, and Weifeng Gao. Energy-Efficient Industrial Internet of Things: Overview and Open Issues. *IEEE Transactions on Industrial Informatics*, 17(11):7225–7237, nov 2021.

[27] Yuyi Mao, Jun Zhang, and Khaled B. Letaief. Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, dec 2016.

[28] Abbas Mehrabi, Matti Siekkinen, Teemu Kämäräinen, and Antti yl¨ J¨¨ski. Multi-tier cloudvr: Leveraging edge computing in remote rendered virtual reality. *ACM Trans. Multimedia Comput. Commun. Appl.*, 17(2), may 2021.

[29] Joao Morais, Sjors Braam, Remco Litjens, Sandra Kizhakkekundil, and Hans Van Den Berg. Performance Modelling and Assessment for Social VR Conference Applications in 5G Radio Networks. *International Conference on Wireless and Mobile Computing, Networking and Communications*, 2021-October:225–232, 2021.

[30] University of Twente. Gitab. `https://gitlab.utwente.nl/`. Accessed: 2022-09-25.

[31] Andy Patrizio. Cloud computing costs & comparisons 2022. `https://www.datamation.com/cloud/cloud-costs/`, April 2017. Accessed: 2022-02-08.

[32] Mittal K. Pedhadiya, Rakesh Kumar Jha, and Hetal G. Bhatt. Device to device communication: A survey. *Journal of Network and Computer Applications*, 129:71–89, mar 2019.

[33] Hans Peters. *Game theory: A Multi-leveled approach*. Springer, 2015.

[34] Larry L Peterson and Bruce S Davie. *Computer Networks: a Systems Approach*. Elsevier, 2012.

[35] Christoph Prager. Why you should have a healthy obsession with startup time. YouTube at `https://bitmovin.com/importance-video-startup-time/`, 2020. Accessed: 2022-09-15.

[36] Qualcomm. Snapdragon 835 mobile pc platform. `https://www.qualcomm.com/products/application/mobile-computing/snapdragon-8-series-mobile-compute-platforms/snapdragon-835-mobile-pc-platform`. Accessed: 2022-08-25.

[37] Kjetil Raaen and Ivar Kjellmo. Measuring latency in virtual reality systems. pages 457–462, 09 2015.

[38] T Bheemarjuna Reddy, John P John, and C Siva Ram Murthy. Providing mac qos for multimedia traffic in 802.11 e based multi-hop ad hoc wireless networks. *Computer Networks*, 51(1):153–176, 2007.

[39] Demóstenes Z Rodriguez, Renata L Rosa, Rodrigo D Nunes, and Emmanuel T Affonso. Assessment of quality-of-experience in telecommunication services. *Int J Digit Info Wireless Commun (IJDIWC)*, 6(4):241–259, 2016.

[40] Jinjia Ruan and Dongliang Xie. A survey on qoe-oriented vr video streaming: Some research issues and challenges. *Electronics*, 10(17):2155, 2021.

[41] Mansoor Shafi, Andreas F. Molisch, Peter J. Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017.

[42] Muhammad Shahid Anwar, Jing Wang, Sadique Ahmad, Asad Ullah, Wahab Khan, and Zesong Fei. Evaluating the factors affecting qoe of 360-degree videos and cybersickness levels predictions in virtual reality. *Electronics*, 9(9), 2020.

[43] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A Survey on Mobile Augmented Reality with 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Communications Surveys and Tutorials*, 23(2):1160–1192, apr 2021.

[44] Xuejing Sun. Immersive audio, capture, transport, and rendering: a review. *APSIPA Transactions on Signal and Information Processing*, 10:e13, 2021.

[45] Zhenfeng Sun, Ming Zhao, and Mohammad Reza Nakhai. Computation offloading in energy harvesting powered mec network. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[46] Gordon Tao, Bernie Garrett, Tarnia Taverner, Elliott Cordingley, and Crystal Sun. Immersive virtual reality health games: a narrative review of game design. *Journal of NeuroEngineering and Rehabilitation*, 18(1):1–21, 2021.

[47] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

[48] Ziling Wei, Baokang Zhao, Jinshu Su, and Xicheng Lu. Dynamic edge computation offloading for internet of things with energy harvesting: A learning method. *IEEE Internet of Things Journal*, 6(3):4436–4447, jun 2019.

[49] Tian Zhang and Wei Chen. Computation Offloading in Energy Harvesting aided Heterogeneous Mobile Edge Computing. *IEEE Vehicular Technology Conference*, 2021-April, apr 2021.

[50] Fengjun Zhao, Ying Chen, Yongchao Zhang, Zhiyong Liu, and Xin Chen. Dynamic Offloading and Resource Scheduling for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE Transactions on Network and Service Management*, 18(2):2154–2165, jun 2021.

APPENDIX A

Surge in offloading-related literature

Computation offloading has already been a subject of numerous studies over the past decades. On scopus, Elsevier's "abstract and citation database" [12], the first paper studying "computation offloading to save energy on handheld devices" [22] dates back to 2001. In their survey, Akherfi et al. note a progressive increase in research works concerning offloading: less than 10 in 2004 cited "computational offloading" while around 40 cited it in 2014 [2]. Ghobaei-Arani et al. also noted an increase in research papers between 2014 and 2018 [13, Figure 2]. By querying scopus [12], the above trends can be replicated. Note that the exact numbers have not been verified in-depth (e.g. search might not be complete) and have to be taken with a grain of salt.

Figure A.1 shows per year how many hits are received when querying "computation offloading"-like terms in their title, abstract or keywords and how many papers thereof *also* cite edge or fog-related concepts. In 2016, 31 of 223 hits (13.9%) study offloading in MEC-enabled scenarios, while in 2021, there are 1004 of 1179 (85.2%) with these criteria. This not only shows the increasing importance of offloading itself but also the role of the edge (or fog) in the development. Moreover, these numbers show that MEC-enabled offloading is an emerging technology and is actively researched at the moment. Another significant trend, namely the rise of machine learning, is depicted in Figure A.2. In 2021, 312 of the 1004 (31.1%) search results also have keywords related to deep learning.

The following search queries have been used when creating Figures A.1 and A.2. Table A.1 shows a tabular overview of the numbers.

```
( TITLE-ABS-KEY ( "computation offloading" )  OR  TITLE-ABS-KEY ( "data
    offloading" )  OR  TITLE-ABS-KEY ( "computational offloading" )  OR
    TITLE-ABS-KEY ( "task offloading" )  OR  TITLE-ABS-KEY ( "offloading
    decision problem" ) )  AND  ( LIMIT-TO ( SUBJAREA ,  "COMP" )  OR
    LIMIT-TO ( SUBJAREA ,  "ENGI" )  OR  LIMIT-TO ( SUBJAREA ,  "MATH" )
    OR  LIMIT-TO ( SUBJAREA ,  "ENER" )  OR  LIMIT-TO ( SUBJAREA ,  "DECI"
    ) )  AND  ( LIMIT-TO ( DOCTYPE ,  "cp" )  OR  LIMIT-TO ( DOCTYPE ,  "ar
    " )  OR  LIMIT-TO ( DOCTYPE ,  "cr" )  OR  LIMIT-TO ( DOCTYPE ,  "re" )
      OR  LIMIT-TO ( DOCTYPE ,  "ch" ) )
```

Listing A.1: Scopus search Query: Offloading

```
(TITLE-ABS-KEY("computation offloading") OR TITLE-ABS-KEY("data offloading
    ") OR TITLE-ABS-KEY("computational offloading") OR TITLE-ABS-KEY("task
    offloading") OR TITLE-ABS-KEY("offloading decision problem") OR TITLE-
    ABS-KEY("MEC-ENABLED" OR "MEC-Assisted") ) AND (TITLE-ABS-KEY("MEC") OR
     TITLE-ABS-KEY("fog computing") OR TITLE-ABS-KEY("edge computing") )
    AND ( LIMIT-TO ( SUBJAREA,"COMP" ) OR LIMIT-TO ( SUBJAREA,"ENGI" ) OR
    LIMIT-TO ( SUBJAREA,"MATH" ) OR LIMIT-TO ( SUBJAREA,"ENER" ) OR LIMIT-
    TO ( SUBJAREA,"DECI" ) ) AND ( LIMIT-TO ( DOCTYPE,"cp" ) OR LIMIT-TO (
```

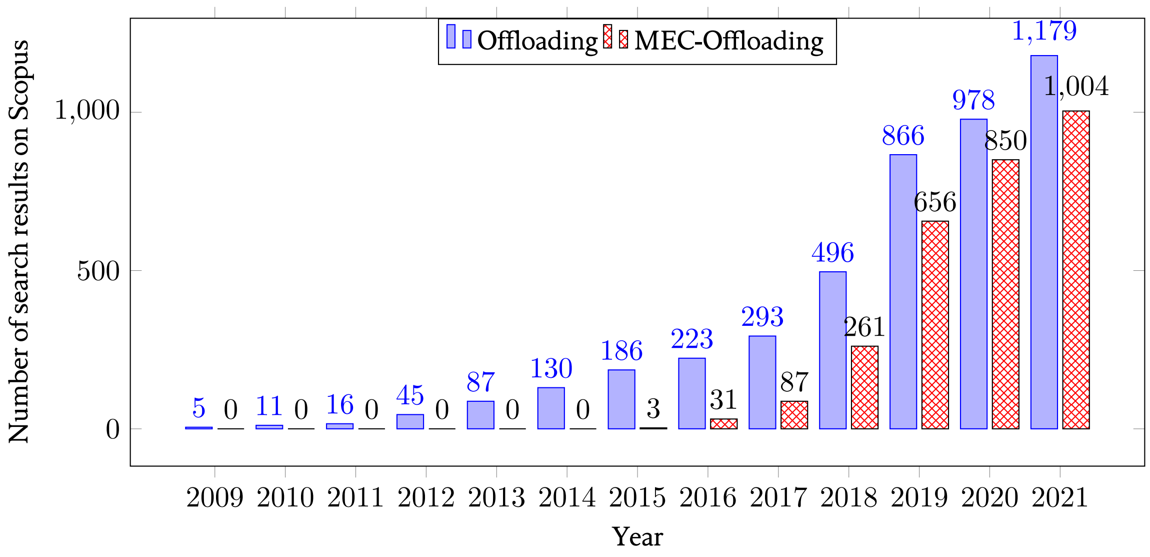


Figure A.1: Number of search results per year on Scopus [12] citing offloading or offloading and MEC, respectively (Listing A.1 and A.2), executed on 21st February 2022, but excluding results from after 2021.

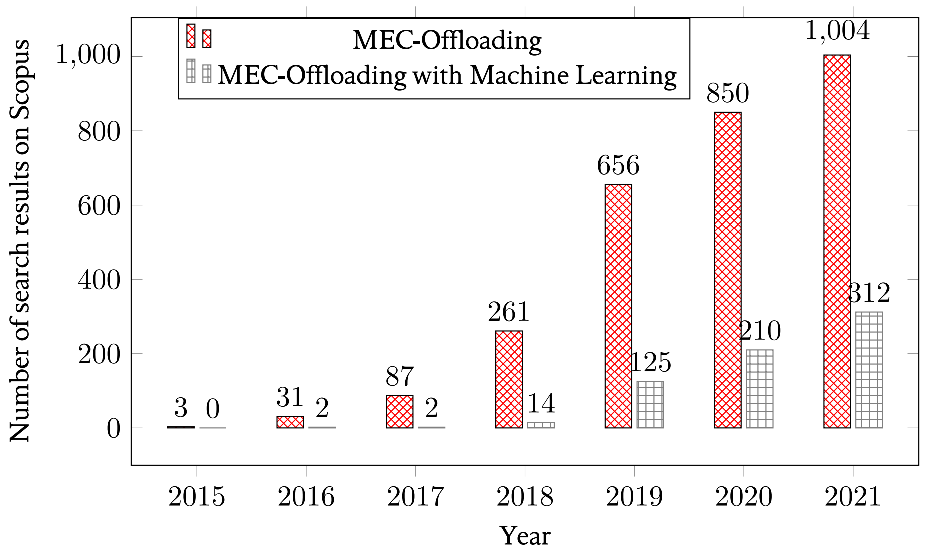


Figure A.2: Number of search results per year on Scopus [12] citing machine learning together with MEC and offloading (Listing A.3), executed on 21st February 2022, but excluding results from after 2021.

```
    DOCTYPE,"ar" ) OR LIMIT-TO ( DOCTYPE,"cr" ) OR LIMIT-TO ( DOCTYPE,"re"
    ) OR LIMIT-TO ( DOCTYPE,"ch" ) )
```

Listing A.2: Scopus search Query: Offloading and MEC

```
(TITLE-ABS-KEY("computation offloading") OR TITLE-ABS-KEY("data offloading
    ") OR TITLE-ABS-KEY("computational offloading") OR TITLE-ABS-KEY("task
    offloading") OR TITLE-ABS-KEY("offloading decision problem") OR TITLE-
    ABS-KEY("MEC-ENABLED" OR "MEC-Assisted") ) AND (TITLE-ABS-KEY("MEC") OR
     TITLE-ABS-KEY("fog computing") OR TITLE-ABS-KEY("edge computing") )
    AND ( LIMIT-TO ( SUBJAREA,"COMP" ) OR LIMIT-TO ( SUBJAREA,"ENGI" ) OR
    LIMIT-TO ( SUBJAREA,"MATH" ) OR LIMIT-TO ( SUBJAREA,"ENER" ) OR LIMIT-
    TO ( SUBJAREA,"DECI" ) ) AND ( LIMIT-TO ( DOCTYPE,"cp" ) OR LIMIT-TO (
    DOCTYPE,"ar" ) OR LIMIT-TO ( DOCTYPE,"cr" ) OR LIMIT-TO ( DOCTYPE,"re"
    ) OR LIMIT-TO ( DOCTYPE,"ch" ) ) AND ( LIMIT-TO ( EXACTKEYWORD,"
    Reinforcement Learning" ) OR LIMIT-TO ( EXACTKEYWORD,"Deep Learning" )
    OR LIMIT-TO ( EXACTKEYWORD,"Learning Algorithms" ) OR LIMIT-TO (
    EXACTKEYWORD,"Deep Reinforcement Learning" ) OR LIMIT-TO ( EXACTKEYWORD
    ,"Genetic Algorithms" ) OR LIMIT-TO ( EXACTKEYWORD,"Machine Learning" )
     OR LIMIT-TO ( EXACTKEYWORD,"Deep Neural Networks" ) )
```

Listing A.3: Scopus search Query: Offloading MEC and Machine Learning

Table A.1: Number of Search Results on Scopus[12] per year and per query (from 21/02/2022)

Year	**Offloading**	**Offloading, MEC**	**Offloading, MEC and Machine Learning**
	Listing A.1	Listing A.2	Listing A.3
2001	3	0	0
2002	1	0	0
2003	2	0	0
2004	3	0	0
2005	4	0	0
2006	4	1	0
2007	4	0	0
2008	10	0	0
2009	5	0	0
2010	11	0	0
2011	16	0	0
2012	45	0	0
2013	87	0	0
2014	130	0	0
2015	186	3	0
2016	223	31	2
2017	293	87	2
2018	496	261	14
2019	866	656	125
2020	978	850	210
2021	1179	1004	312
2022	154	145	45
Total	**4700**	**3038**	**710**

APPENDIX B

Explanations of factors involved in the offloading decision problem

This chapter defines and elaborates on the factors mentioned in Figure 2.4. All details necessary to understand our research are given in Chapter 4. This appendix provides a high-level overview of a more general offloading decision setting. Aspects that were not analyzed in detail in this final project could be starting points for future work.

B.1 System model

First, the physical **system** can take many forms: both considering the overall infrastructure as well as the individual components. Hence, we describe the different entities and properties in detail here below.

1. **(Mobile and Wireless) Network** refers to the transmission capabilities among agents in the system. In our context, the focus is on wireless mobile networks. Ultimately, the network model determines the data rate of users and base stations (BSs), so how much (throughput) and how fast (latency) data can be transmitted.
   In a single system, network specifications can differ per communication link. For example, the *backhaul* link from a BS to the core network could be a reliable, high-speed fibre-optic connection. In contrast, the *fronthaul* connection between BSs could be some custom protocol from the network operator. Similarly, the uplink transmission from end-user to BS can use the 5G sub-6GHz link, while downlink communication uses 5G mmWaves [16].
   In models, digital communication is often abstracted to the following:

   Coverage for wireless connections. What is the transmission range of the Base Station or the end devices? How is the signal strength affected by the **distance** from the source? Modelling the fading (decrease in signal strength between source to destination) is important to determine which devices can successfully exchange transmissions.

   Noise and Interference. To model the achievable data rate, the signal-to-noise-and-interference ratio (SINR) of a user is calculated. SINR indicates the effect of the environment on the signal strength. The lower the effect of noise and interference on the signal strength, the larger the SINR ratio and the better transmission quality. The signal strength depends on the transmission power from the emitting device, while interference highly depends on the transmission power from other devices.

   Bandwidth. In the literal definition, the bandwidth is expressed in Hertz (Hz) and measures the width of a frequency band. In practice, the available bandwidth indicates how much data can maximally be transmitted over a link (in bits per second) [34]. The actual bandwidth available to a user depends on the protocol implementation and bandwidth allocation (how the bandwidth is shared among end-users). The latter often depends on the number of users connected to a BS.

Refer to Table III of [26] for a "comparison between short-range and long-range communication technologies" illustrating the coverage, data rates and typical applications of technologies such as 4G, 5G and LoRa.

2. **Entities** describe all devices connected through the network that participate in the system. From a high-level point of view, we distinguish the *end device* (or *end-user*), such as HMDs, mobile phones, smart cameras and sensors, that run a certain application for which offloading is considered. Then, *edge* and *cloud* entities have the processing capabilities to handle offloaded tasks.

   Number of entities. How many of the different entities are present? Some variations are shown in Figure B.1.

   Heterogeneity. If there are more entities of the same type, are they homogeneous or heterogeneous? In other words, if there are multiple end devices, do they all have the same properties, such as processing power, transmission power or energy consumption (homogeneity)? Or do the devices differ in terms of capabilities (heterogeneity)? Furthermore, if there are multiple offloading decision-makers, they could have different objectives. Figure B.1 shows an example with heterogeneous end devices and edge nodes.

Most offloading schemes consider a selection of the following entities.

1. **Edge or Fog device.** Often seen as part of 5G and beyond network infrastructure, edge devices bring processing power closer to end-users, namely to the network *edge*. It is agreed that *fog* and *edge* both refer to this computing paradigm, but opinions differ on how to distinguish them precisely [19]. We stick to the definition of *edge* as part of the telecommunication perspective (5G and beyond), while *fog* is the more general description of bringing computing resources to end devices. The *edge device* (or *edge node, edge server*) often resides at the BS.

   Processing Power is expressed in CPU cycles per second. It indicates the computational capacity of a device, so how quickly (cycles per second) a task (number of cycles) can be processed. The actual processing power allocated to the task of an end-user depends on how the processing power is divided and shared among all end-users connected to the edge node.

   Memory, Transmission Power, Energy consumption and Source of Energy are explained below under **End Device**.

   Mobility indicates whether the device is moving or stationary. For instance, unmanned aerial vehicles (UAVs) providing communication and offloading services can be considered as moving fog nodes [25].

   Cost. For commercial edge networks, a monetary cost might be associated with the use of computational or communications resources.

   Caching, temporarily storing popular content for quick accessibility at the edge is an important factor in VR applications and, therefore, also needs to be taken into account when offloading in a VR setting. Who decides what to store and why can differ among edge operators.

2. **Cloud Center.** Numerous cloud providers such as IBM SoftLayer, Amazon, Google and Microsoft provide computational resources in large data centres committed to providing high processing power [31]. Elements specific to the inner workings of a cloud centre are out-of-scope, so the factors here below are limited to what could influence the decision to offload a task to the cloud.

   Location. Depending on the data centre's location, the round-trip time of sending tasks to the cloud can outweigh the benefits of the quicker computation time due to the high processing power. Furthermore, the location of the cloud centre influences what information can legally be offloaded. For example, in Europe, transferring personal data to a third country can only happen if strict conditions are fulfilled according to the GDPR [6].

   Cost associated with using the cloud centre's resources in monetary value, often a "pay as you go access" [31].

   Processing power allocated to the offloaded task to model how quickly (cycles per second) a task (number of cycles) can be processed.

3. **End Device.** Properties of the end device influence the functioning of a device, such as transmission and computational strength. The following are the most important properties.

   Memory indicates how much storage a device has available, both in long-term or temporary storage. For example, a device needs long-term storage to save the models needed for the execution of tasks [16] or to keep track of the environment's history for better predictions. Short-term storage, such as RAM, is significant for processing computational tasks that need temporary storage.

   Processing power is expressed in CPU cycles per second and indicates the computational capacity of a device. It affects the processing latency of a task.

   Transmission power determines the signal strength of the emitting device. The higher the transmission power, the larger the range and the better the signal quality is at the destination. However, this goes at the expense of energy consumption.

   Energy consumption indicates the energy cost associated with the processing power or the transmission power, respectively.

   Source of energy. Some systems have access to renewable energy sources through energy harvesting, such that energy is not a limiting factor but should be utilized if available [27]. Other systems that also rely on batteries and backup batteries require a more careful design concerning energy consumption [24]. Depending on the energy source, the energy cost and processing capabilities might differ [24].

   Mobility indicates whether end devices are stationary or mobile. If they are mobile, one must consider the speed, paths travelled (pre-defined or in any direction), predictability (random or scheduled movement) and covered distance (same cellular cell or inter-cell movement). The position of an end device (and, therefore, the relative distance to BSs) affects the SINR. Furthermore, when end devices move out of range from a BS during a time period, handover procedures (transferring the responsibility to communicate with a device to another Base Station) and fault tolerance of the application (how to handle errors, such as tasks being dropped) need to be considered.

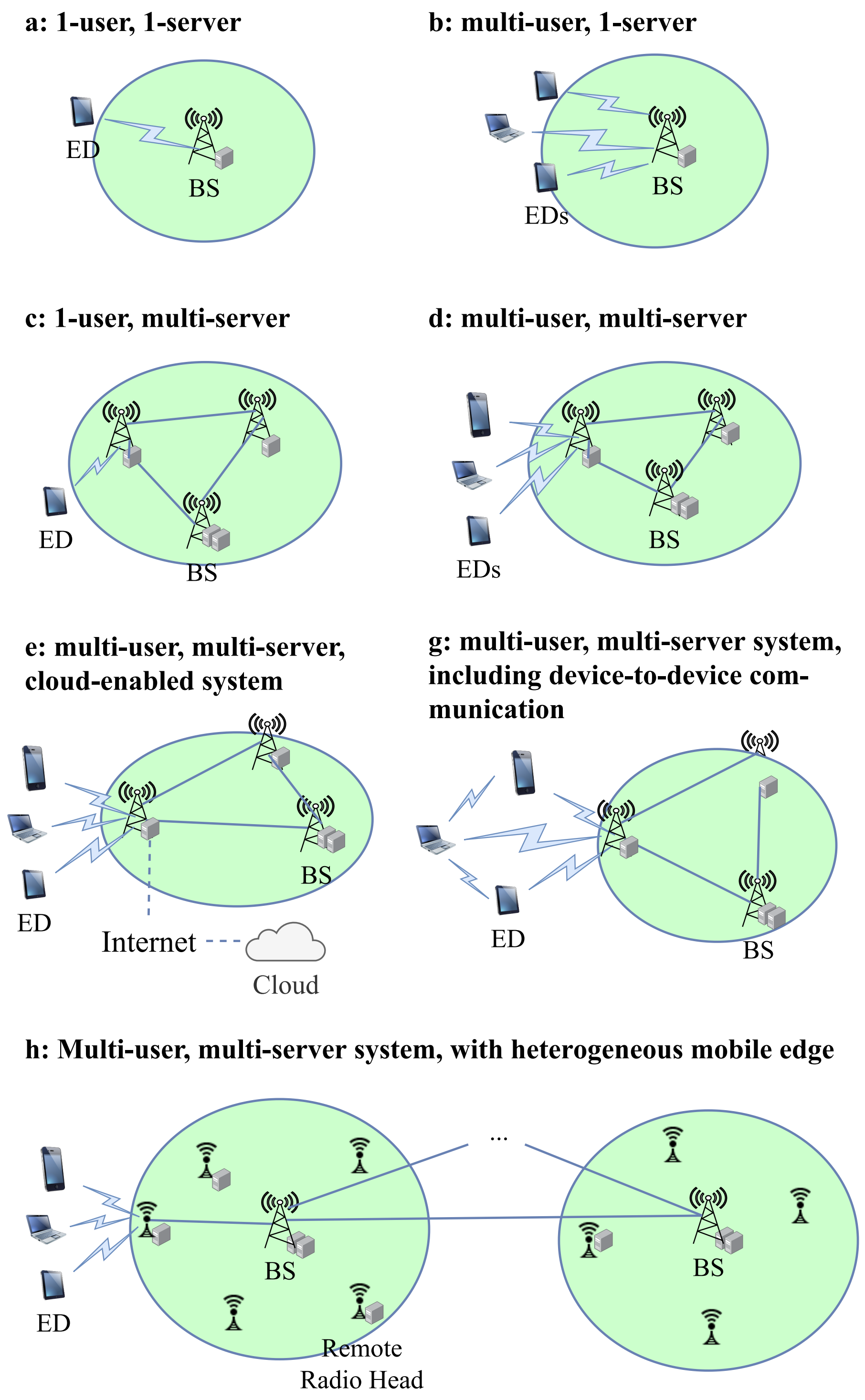

a: 1-user, 1-server
ED
BS
b: multi-user, 1-server
EDs
BS
c: 1-user, multi-server
ED
BS
d: multi-user, multi-server
EDs
BS
e: multi-user, multi-server, cloud-enabled system
ED
BS
Internet
Cloud
g: multi-user, multi-server system, including device-to-device communication
ED
BS
h: Multi-user, multi-server system, with heterogeneous mobile edge
ED
BS
Remote Radio Head
...
BS

Figure B.1: System variants

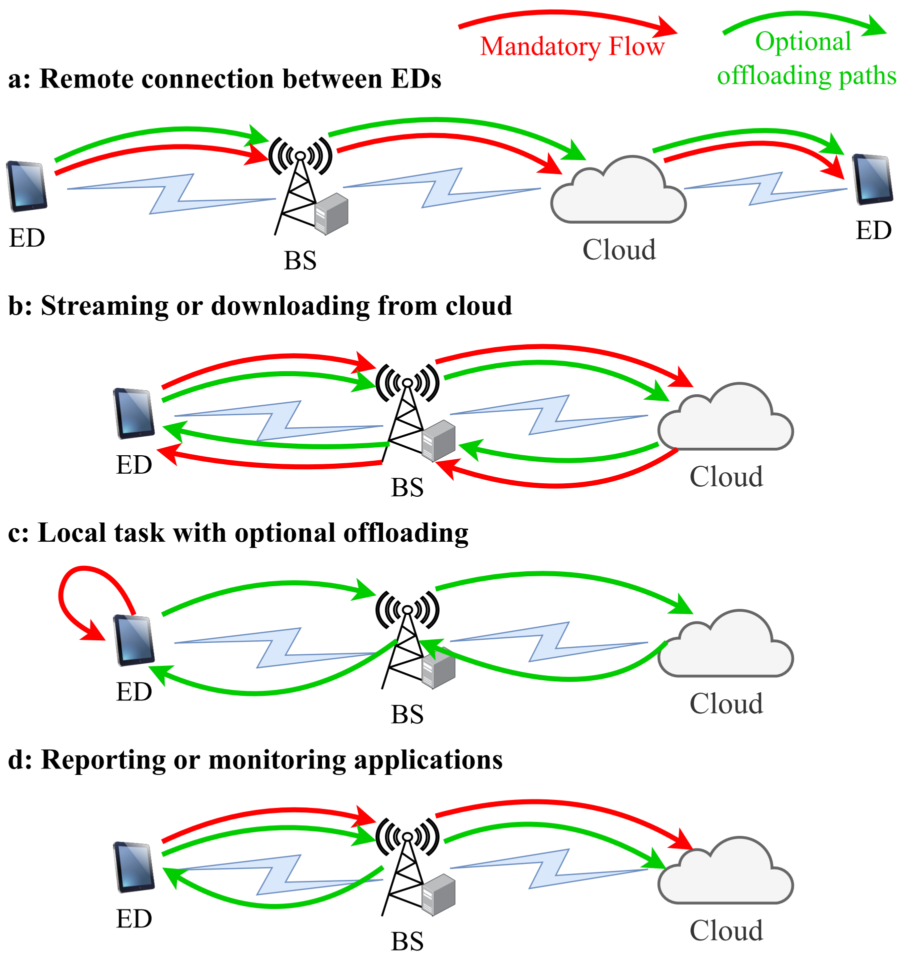

Mandatory Flow
Optional offloading paths
a: Remote connection between EDs
ED
BS
Cloud
ED
b: Streaming or downloading from cloud
ED
BS
Cloud
c: Local task with optional offloading
ED
BS
Cloud
d: Reporting or monitoring applications
ED
BS
Cloud

Figure B.2: Different information flows.

B.2 Application requirements

Whether or not to offload an application's task highly depends on its characteristics and requirements, summarised as follows.

1. **Type.** Depending on how an application is subdivided into and generates different tasks, it is suitable for offloading or not. It encompasses the following properties.

   Granularity relates to the 'unit' of the task considered for offloading: is the task a full program or a simpler piece of code that needs processing, or is the task mainly comprised of data that needs to be preprocessed?

   Flow describes the mandatory and optional flow of information of the application. Variants are shown in Figure B.2.

   Arrival rate. The arrival rate of computational tasks defines how many tasks arrive to be processed in a time period.

   Partitioning defines whether a task can be subdivided into smaller tasks. I.e. can a task be *partially* offloaded, or is the decision either *fully* offload everything or nothing?

   Prioritization defines which tasks of an application should be handled with a higher urgency than others. For example, Reddy et al. improved the transmission of video files by prioritizing I-Frames containing the base picture [38].

2. **Computational Requirements.** Which requirements or constraints play a *physical* role in how a task can be computed? They are distinguished from the *quality* requirements of a task mentioned below.

   Dependencies. If multiple tasks have to be computed, can they be executed in parallel or do inputs to certain tasks depend on outputs of other tasks?

   Prerequisites. Whether or not the model accommodates tasks that need specific circumstances and therefore might not be executable at every destination.
   For example, the VR rendering task converts models into actual pictures [16], so it requires these models to be present on any system before executing the task. Another example is privacy-sensitive tasks that need to be processed locally to prevent malicious interference during transmissions.

   Size. In offloading models, the size of a task is often expressed in CPU cycles per second necessary to process it.

3. **Quality Requirements.** Quality requirements are requirements related to how well the application is functioning and not to the practical computation of the task. This includes the effects on user experience.

   Quality of Service / Experience (QoS/QoE) requirements dictate the minimum performance required for a smooth operation of the application. QoE refers to the overall user experience with the application, while QoS are more technical, service-level agreements guaranteeing network performance [39]. Examples of QoS requirements are minimum delay or bandwidth, or maximal jitter and packet loss rate specifications, and examples of QoE are latency, availability and fault tolerance [1, 39].

   Security and Privacy. Confidentiality (no eavesdropping), Integrity (no tempering) and Availability (no outage), denoted by CIA, are well-known security requirements. Other security concepts include authentication (identity is verified), non-Repudiation (authorship cannot be refuted) and authorization (actions are permitted) [32], as well as privacy and data protection [19].

B.3 Offloading decision problem

The offloading decision problem is the core of resource management involving devices with different processing capabilities. Considering the given system and all the application's requirements, deciding whether or not to offload computations to another device is the key problem to solve. Despite identical system and application models, the offloading decision problem can still differ depending on the following factors, increasing the complexity.

1. **Decision moment.** When is the decision on whether and how to offload being made? Is it a pre-defined percentage [4], (static decision) or is it a dynamic decision being adapted in every time slot? [16] Is the decision reactive to network changes or proactive?

2. **Objective.** The objective defines *what* (performance) metric should be optimized in the offloading process and *for whom*. For example, minimizing latency [49], minimizing overall energy consumption for edge servers and end devices [48] or only minimizing energy consumption of the end devices [50].

3. **Constraints** define restrictions and checks that the system should always uphold during the optimization. A system constraint could be the caching limit of an edge server, while a performance constraint could be an upper bound for the latency. In other words, even if it would optimize the energy consumption objective, a trade-off leading to a latency above this threshold cannot be considered.
4. **(Offloading) destinations** are the places or devices where a task could potentially be processed. In MEC-enabled dynamic offloading models, this, typically, at least includes local processing or computing at the edge. Other possibilities are: dropping the task, processing in the cloud and offloading to another end device.
5. **Actions.** The decision problem intends to optimize the objective by choosing the best available action. In the offloading decision problem, the action includes the chosen offloading destination. In joint optimization schemes, actions can, for example, also include the server selection [45] or caching policy [16].
6. **Decision Maker.** Which entity makes the offloading decision? This simple concept can become quite complex because it ties into the information available to the decision-maker. If the decision is made centrally, the central entity needs to gather all the information necessary, which can lead to overhead if many entities are part of the system. If the decision is distributed - e.g. every end device makes its own offloading decision, bottlenecks risk arising. For example, if every device decides to offload to one specific edge server, its processing capabilities risk being exceeded.
7. **Mathematical technique.** Li et al. provide an extensive summary of the most common techniques to model offloading by elaborating on definitions, differences, advantages and disadvantages and by giving examples of offloading schemes based on the different techniques [23]. The following enumeration briefly summarizes the techniques they describe.

   Convex and non-convex optimization techniques such as linear (convex) and mixed-integer (non-convex) optimization problems are classic mathematical optimization techniques that analytically seek to find the action(s) that optimize some objective function under pre-defined system constraints. While such techniques are mathematically proven to provide local or global optima, they often cannot handle highly complex situations and are unable to take into account a changing environment.

   Lyapunov optimization is used to optimize dynamical systems that use network queues by defining a "drift-plus-penalty-expression" [23]. For example, application tasks or energy processes can be modelled using *queues* and be represented in the *drift*, whereas the objective is embodied in the *penalty*. By jointly minimizing drift and penalty, an optimal solution is sought. The advantages of this method are that their solutions have lower computational complexity than those of (non)-convex optimizations while still being close to optimal. Disadvantages include that the optimum cannot be guaranteed, and some unrealistic modelling assumptions need to be addressed, such as time-independent and identically distributed action sets.

   Markov Decision Processes (MDP) can model *dynamic* decision making in an uncertain and varying environment by defining states and seeking an optimal decision for every state. With state-dependent actions and transition probabilities (chance of transitioning from one state to another given some action), uncertainties in the system can be modelled. A disadvantage is that MDPs suffer from the *curse of dimensionality*: the more complex systems, the more states are required to model the system properly, and an algorithm's computing times might increase exponentially as a consequence. Another disadvantage is that many real-life scenarios do not have well-known and well-defined transition probabilities.

Game Theory (GT) is a powerful mathematical technique to model the (distributed) decision making and resulting interaction of multiple *players* (or users) of a system, each player trying to optimize their own benefit. The solution, called *Nash Equilibrium*, to such a system is a state in which no player can switch their action and receive a higher reward if all other players keep their original action. The advantage is that it allows for the modelling of distributed, independent and selfish decision-making of multiple actors. Furthermore, stochastic games can also consider an uncertain and varying environment. The main disadvantages are that a Nash Equilibria might not be global optima and that, similarly to MDPs, stochastic games might also suffer from the curse of dimensionality.

Machine learning (ML) defines self-learning algorithms that seek a solution by learning from past behaviours and then predicting the future. A form of machine learning suitable for modelling offloading is (deep) reinforcement learning (DRL) since it is based on MDPs. Because of its ability to learn, it is often applied to stochastic environments where transition probabilities are hard to model or even unknown. A comprehensive survey on deep learning in mobile edge is done by [47]. By using detailed schemes and tables, they classify the use of deep learning, the different algorithms' advantages and disadvantages, typical application areas and focus on the difference between "intelligent edge" and "edge intelligence". The former relates to deep learning embedded in the functioning of the edge (such as using machine learning in the offloading decision problem), while the latter refers to intelligent applications (e.g. self-driving cars or facial recognition) making use of the edge infrastructure.

Considerations to decide which modelling technique is appropriate for a scenario include: is it a centralized (one party deciding for all actors) or a distributed (every entity making their own choice) decision? Is the scenario static (system parameters do not change) or dynamic (an uncertain environment is involved)? How can information be distributed in the system to determine what information would be available for the decision-maker(s)? Does the model need to scale? Are the entities powerful enough to calculate solutions to the model? [23]

In addition, we make a distinction on whether the model can be solved **analytically** or using **simulations.**

Analytically. Some traditional optimization techniques can be evaluated analytically and guarantee the optimal solution. However, the complexity and uncertainty inherent to the offloading problem make analytical solutions hard or even impossible: they often require significant simplifications of the models (e.g., making a system linear) so that they are no longer feasible in practice.

Simulations. Most studies evaluate their offloading scheme using simulations. By implementing the system, application and offloading decision scheme and simulating a stochastic environment, they can get a more realistic insight into the performance of a scheme. However, due to a large number of factors, comparing the simulation results between different offloading schemes is often unjust. Commonly used simulation frameworks include iFogSim (30%), CloudSim (19%), MATLAB (15%), and Java JMT (9%) according to [13]. Studies that make use of machine learning solutions often evaluate their scheme using TensorFlow [16].

APPENDIX C

Analysis of Guo et al.'s offloading scheme: recapitulation of their formulas

C.1 Guo et al.'s computing model

The rendering delays in Guo et al.'s rendering scheme are modelled as follows. Note that these formulas are taken from [16]. The only alterations are changing the variables such that they are consistent with the remainder of the thesis and substituting the calculations of the SINR by a fixed value γ.

Table C.1: Table of simulation parameters by Guo et al. The parameters in **bold** differ from our simulation parameters.

Symbol	Name	Value	Unit
b_{z_c}, b_{z_l}	Number of bits processed in a CPU cycle at MECs or locally	0.4	bits/CPU cycle
-	Uplink data ratio (ratio that needs to be communicated uplink)	1/1000	-
Z_C	Total processing power of the MECs	1000e9	Hz
z_l	Local processing power of the HMD	1e9	Hz
B^u	Uplink bandwidth	10	MHz
B^d	Downlink bandwidth	**1000**	MHz
q_i	Quality resolution	$\{720, 600, 480, 720\}$	p
C_f	Computational requirement of the frame's foreground	$\{2.5e6, 5e6, 10e6, 20e6\}$	CPU cycles
C_b	Computational requirement of the frame's background	$\{10e6, 20e6, 40e6, 60e6\}$	CPU cycles
α	*Number of people associated to the base station (used in Chapter 3, but not by Guo et al.)*	**5**	user

The simulation parameters of Guo et al. are summarised in Table C.1 and Guo et al. provide the following formulas to calculate the data sizes:

$$h(q_i) = \frac{5}{9}q_i^2,$$
$$D_f = h(q_i)/1000,$$
$$f(q_0) = 10\text{MB},$$
$$g(q_0) = 2\text{MB}.$$

where q_i is the resolution. Looking at these formulas, it is unclear to determine their derivation: 5/9 probably results from assuming a 5:9 screen resolution and then having q_i pixels as length and $5/9q_i$ pixels as height resulting in a length times height number of pixels.

The data rate of a user R is calculated as follows, based on the bandwidth B, the number of people associated with the base station α and the SINR value γ:

$$R = \frac{B}{\alpha} \cdot \log_2(1+\gamma).$$

Note that they calculate with 100 users and 10 base stations, but we simplify this to the number of users α that are associated with the base station, because in Chapter 3 we assume the point of view of a single user.

In *local rendering*, both fore- and background of a frame are rendered locally at the VR device. No communication to the base station is needed. The latency for local rendering consists of the delay to render the back- and foreground with computational requirements C_b and C_f on the local device with processing power z_l.

$$T^{local} = \underbrace{\frac{C_b + C_f}{z_l}}_{\text{Render back- and foreground}} \tag{C.1}$$

In *remote rendering*, both fore- and background of a frame are rendered remotely at the base station. For the base station to render the foreground, it needs to receive some information from the VR device. It is assumed that this information is not needed to render the background.

The latency for remote rendering consists of the delay for the uplink transmission, the rendering of the frame (depending on whether the background is cached or not), the delay to compress the frame for transmission, the downlink transmission delay of the compressed frame and decoding delay to decompress the frame containing fore- and background information:

$$\begin{aligned} T^{remote} = & \underbrace{\frac{D_f}{R^u}}_{\text{Uplink transmission}} + \underbrace{\frac{C_f}{z_c} + \begin{cases} \frac{C_b}{z_c}, & \text{if not cached} \\ \frac{k(q_i)}{z_c \cdot b_{z_c}}, & \text{if cached} \end{cases}}_{\text{render fore- and background}} + \underbrace{\frac{f(q_0)}{z_c \cdot b_{z_c}}}_{\text{Compression latency}} \\ & + \underbrace{\frac{g(q_0)}{R^d}}_{\text{Downlink Transmission}} + \underbrace{\frac{h(q_i)}{z_l \cdot b_{z_l}}}_{\text{Decoding compressed frame}} , \end{aligned} \tag{C.2}$$

where D_f is the uplink data size, R^u and R^d the up- and downlink data rates, C_f and C_b the fore- and background computational requirements, $k(q_i)$

In *cooperative rendering*, the foreground of a frame is rendered locally at the VR end device, while the background is rendered at the base station. No uplink communication is needed.

$$T_{coop} = \max\Bigg(\underbrace{\frac{C_f}{z_l}}_{\text{local foreground rendering}},$$

$$\underbrace{\underbrace{\begin{cases} 0, & \text{if cached} \\ \frac{C_b}{z_c}, & \text{if not cached} \end{cases}}_{\text{Remote background rendering}} + \underbrace{\frac{f(q_0)}{z_c \cdot b_{z_c}}}_{\text{Compression latency}} + \underbrace{\frac{g(q_0)}{R^d}}_{\text{Downlink Transmission}} + \underbrace{\frac{h(q_i)}{z_l \cdot b_{z_l}}}_{\text{Decoding compressed frame}} \Bigg)}_{\text{Remote background rendering}}$$

$$+ \underbrace{\frac{k(q_i)}{z_l \cdot b_{z_l}}}_{\text{integrate fore- and background}} \tag{C.3}$$

C.2 Maximum number of users based on Guo et al.'s computing model

Based on the formulas in the previous section, the maximum number of people that can simultaneously render remotely (under the same system state) can be calculated as follows. Note that this formula assumes that everyone renders frames with the same quality resolution q and computational requirements C_f and C_b. The upper bound on the number of users that can be associated with the same base station j as user i for user i to meet its latency requirement is calculated as follows. To include the two cases (the background frame is cached, or it is not cached), we define $v \in \{0, 1\}$ as the caching parameter, $v = 1$ if the background is cached and $v = 0$ if not. Let N be the number of users associated with base station j (including user i). Then the following values depend on N:

$$\begin{aligned} R^u &= \frac{B^u}{N} \cdot \log_2(1+\gamma) \\ R^d &= \frac{B^d}{N} \cdot \log_2(1+\gamma) \\ z_c &= \frac{Z_c}{N}. \end{aligned}$$

Filling this into the equation for T^{remote}:

$$\begin{aligned} &\frac{D_f}{\frac{B^u}{N} \cdot \log_2(1+\text{SINR}^u_{ij})} + \frac{C_f}{\frac{Z_c}{N}} + \frac{1}{\frac{Z_c}{N}} \cdot \frac{(1-v)C_b b_{z_c} + v k(q_i)}{b_{z_c}} + \frac{f(q_0)}{\frac{Z_c}{N} b_{z_c}} \\ &+ \frac{g(q_0)}{\frac{B^d}{N} \cdot \log_2(1+\text{SINR}^d_{ij})} + \frac{h(q_i)}{z_l \cdot b_{z_l}} \le \tau \\ \iff &N \cdot \underbrace{\left(\frac{D_f}{B^u \cdot \log_2(1+\text{SINR}^u_{ij})} + \frac{C_f}{Z_c} + \frac{(1-v)C_b b_{z_c} + v k(q_i)}{Z_c b_{z_c}} + \frac{f(q_0)}{Z_c b_{z_c}} + \frac{g(q_0)}{B^d \cdot \log_2(1+\text{SINR}^d_{ij})} \right)}_{\ge 0} \\ &\le \tau - \frac{h(q_i)}{z_l \cdot b_{z_l}} \end{aligned}$$

Since all individual terms and factors in the coefficient of N are positive, we can divide by the coefficient and determine the upper bound for N:

$$N \leq \left(\tau - \frac{h(q_i)}{z_l b_{z_l}}\right) \cdot \frac{1}{\frac{D_f}{B^u \cdot \log_2(1+\mathrm{SINR}_{ij}^u)} + \frac{C_f}{Z_c} + \frac{(1-v)C_b b_{z_c} + vk(q_i)}{Z_c b_{z_c}} + \frac{f(q_0)}{Z_c b_{z_c}} + \frac{g(q_0)}{B^d \cdot \log_2(1+\mathrm{SINR}_{ij}^d)}}.$$

Figure C.1 shows the maximum number of users that can simultaneously offload for different parameters.

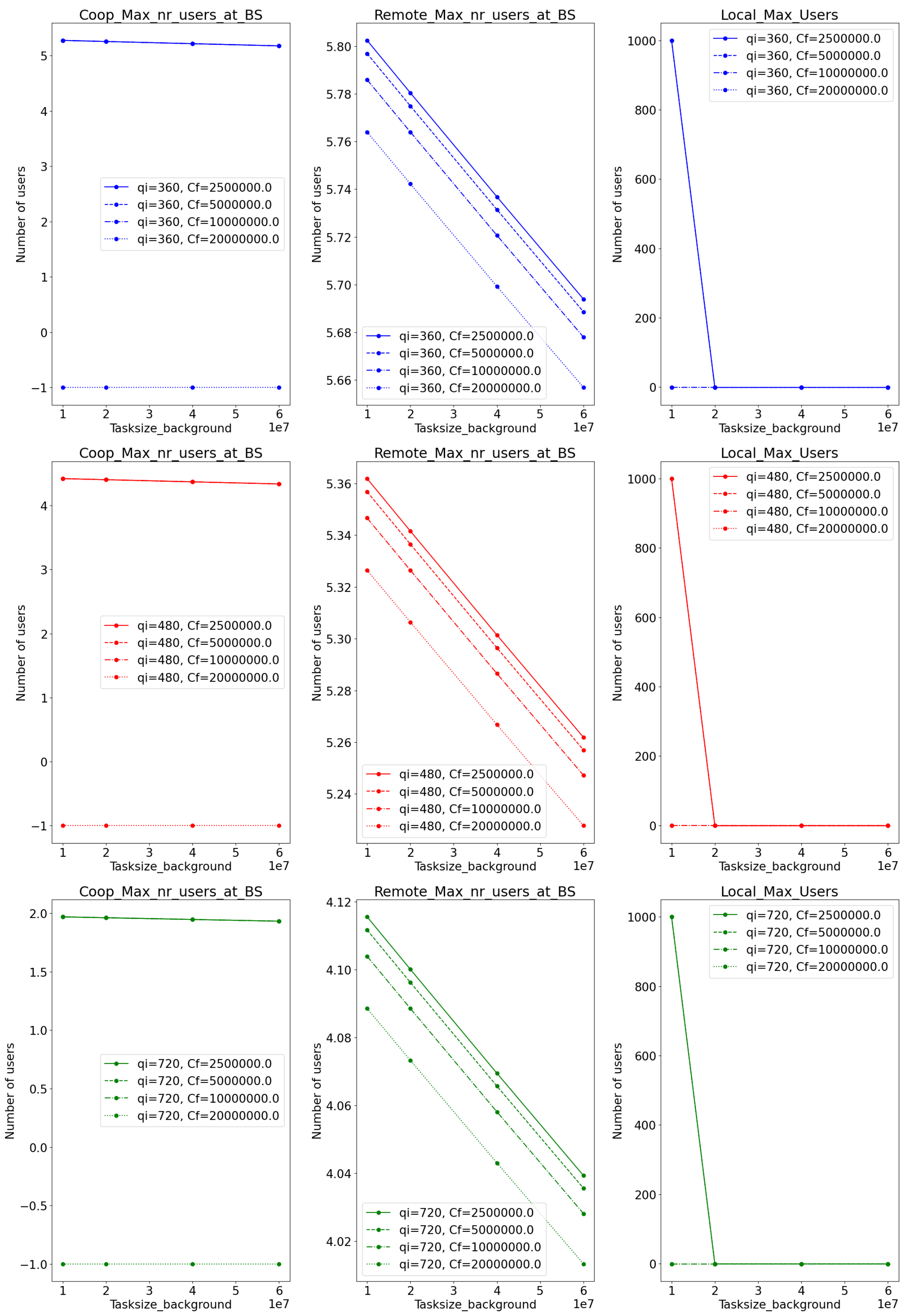


Figure C.1: Maximum number of people that can simultaneously offload under different parameters when $\theta_t = 1/60$ s and $\gamma = 20$ dB.