Performance Study of Mixed Reality for Edge Computing

Klervie Toczé
klervie.tocze@liu.se
Linköping University
Linköping, Sweden

Johan Lindqvist
johli392@student.liu.se
Linköping University
Linköping, Sweden

Simin Nadjm-Tehrani
simin.nadjm-tehrani@liu.se
Linköping University
Linköping, Sweden

ABSTRACT

Edge computing is a recent paradigm where computing resources are placed close to the user, at the edge of the network. This is a promising enabler for applications that are too resource-intensive to be run on an end device, but at the same time require too low latency to be run in a cloud, such as for example mixed reality (MR).

In this work, we present MR-Leo, a prototype for creating an MR-enhanced video stream. It enables offloading of the point cloud creation and graphic rendering at the edge. We study the performance of the prototype with regards to latency and throughput in five different configurations with different alternatives for the transport protocol, the video compression format and the end/edge devices used.

The evaluations show that UDP and MJPEG are good candidates for achieving acceptable latency and that the design of the communication protocol is critical for offloading video stream analysis to the edge.

CCS CONCEPTS

• **Networks → Network performance analysis**; • **Human-centered computing → Empirical studies in ubiquitous and mobile computing**; • **Computer systems organization → Cloud computing**; *Real-time systems.*

KEYWORDS

Edge/fog computing; mixed reality; open-source prototype; empirical performance evaluation.

ACM Reference Format:
Klervie Toczé, Johan Lindqvist, and Simin Nadjm-Tehrani. 2019. Performance Study of Mixed Reality for Edge Computing. In *Proceedings of the IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC '19), December 2–5, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3344341.3368816

1 INTRODUCTION

In the last decade, the rise of user resource demand for applications has led to the emergence of cloud computing, that is especially heavily used for providing storage as a resource. With the advent of the Internet of Things (IoT), the amount of data to be processed is straining networks. Moreover, new application areas such as online gaming or remote surgery have stringent latency requirements that the cloud, due to its typically remote location, cannot provide.

Therefore, a new paradigm has appeared where computation and storage resources are moved closer to the end user, as a complement to the cloud computing infrastructure. Fog computing [7], edge computing [23] or distributed cloud bears with it promises such as low latency, increased privacy and reduced load on the network. In the rest of the paper, we choose to use the term edge computing to refer to this new paradigm.

Application areas for edge computing are very diverse, including content delivery, healthcare, connected vehicles, and smart grid, and adopt the paradigm in various degrees of maturity[1]. However, it is hard to find data coming from a real implementation in some application area in order to make systematic performance benchmarking studies. This is one motivation for choosing mixed reality (MR) as a vehicle for such studies in this paper.

What does this MR area correspond to? In a recent survey, Bekele et al. [5] present the reality-virtuality continuum and its evolution since the original description by Milgram and Kishino [18]. This continuum presents the span between reality on one side and virtual reality (VR) on the other side. In between are augmented reality (AR) and augmented virtuality (AV), depending on how much virtuality is embedded in the real-world. MR is the umbrella term for both AR and AV, designating any technology mixing reality and virtuality.

MR is an application area that is well-suited for edge computing since it usually requires a high amount of computation and low latency in order to achieve high quality of service (QoS). In this study, we consider an MR application that constructs a 3D map of the user environment and can place virtual objects into it. Such an application requires advanced computer vision algorithms in order to perform the task. Our goal is to study the case where such an application would be run on a device with limited capabilities, and understand trade-off between various choices from the end device to the edge and back. To the best of our knowledge, there is no freely available MR application using edge computing that can be used to understand the resource-timeliness trade-offs in this area.

We believe that this knowledge is useful for several reasons. First, it is of interest for all researchers designing algorithms and techniques for the edge paradigm (e.g. task placement), to have information about as many workloads as possible in order to test their work. Further, it gives some insights for technology developers about the challenges to overcome in order to deploy such applications and highlights areas for conducting further research.

Hence our work aims at answering the following research questions: Is offloading all MR-related computations to the edge empirically feasible? What resources are required (especially for networking, but also for computing on the devices)? What is the impact on

UCC '19, December 2–5, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6894-0/19/12…$15.00
https://doi.org/10.1145/3344341.3368816

performance considering different design choices for specific parts of the system?

Those research questions are answered by the following contributions: 1) MR-Leo, a generic framework for offloading analysis of video streams to the edge, 2) a prototype implementation of it in the context of MR, released open-source, 3) a study of the impact of current one-hop communication on the overall performance, and 4) a study dissecting the end-to-end latency of one frame on a couple of hardware configurations. Essentially our study points clearly to the significance of reducing transmission times for this class of applications, e.g. through future generation networks.

The paper is structured as follows. Section 2 presents related works. Section 3 introduces offloading for the edge and our case study. Then, Section 4 presents the MR-Leo implementation, and Section 5 describes the performed performance evaluation. Results with regards to communication are in Section 6 and we dive into end-to-end frame latency in Section 7. We conclude in Section 8.

2 RELATED WORKS

In a recent survey, Chatzopoulos et al. [9] describe the current state of the mobile AR area and the need for offloading some of the heavy computations required by AR applications. Such an offloading can be done: on a companion device, in the cloud or at the edge; and in two ways: either the rendering is still performed locally or both the rendering and the computing are performed on the remote device. Another possibility is to use approximate computing but this technique is still at an early stage. In this paper, we investigate the case where both rendering and computing are offloaded.

The idea of offloading MR tasks has been studied in the context of cloud computing. Luo [17] implemented a gesture interface running on a local cloud located within the University of Illinois that is rendered on a display wall. Naqvi et al. [21] implemented an enhanced indoor positioning system giving the user extra information about where she is by analyzing the environment. They successfully offloaded the object recognition task to the cloud. However, the recognition latency was high, at best 2 seconds. Shea et al. [25] created an application similar to the Pokemon Go game where they offload part of the computation to the cloud. The rendering is performed in the end device, which differs from our study. Huang et al. [15] propose a framework called CloudRidAR for creating AR applications using the cloud for offloading parts of the AR tasks. In their prototype, the rendering is done on the end device and the feature tracking at the edge as this part is more computation-heavy.

As edge computing aims at reducing application latency, it is promising for MR and a few works have started investigating it. For example, Ahn et al. [3] study the filtering of AR application content using reinforcement learning techniques. They evaluate their proposed optimized policy in a custom-built AR simulator based on the Unity game engine. The same group made a later study [2] about imitation learning for personalized AR, still using their custom AR simulator. Different from them, we implement and evaluate an actual implementation of an MR application. Zhang et al. [31] study the use of different edge devices for performing the tasks of encoding and rendering, in order to improve quality of service. They evaluated their optimized solution with simulated user demand. Our work is complementary, as we study performance with actual user demand, for one edge device computing both the rendering and the encoding.

In their work, Chen et al. [10] study prototypes for seven different wearable cognitive assistance applications. They use their Gabriel platform [14] for implementation. Their evaluation study includes a comparison of running the application at the edge or in the cloud, using 4G or WiFi for the first hop connection, and a comparison of using smart glasses and mobile phones. Their study shares some similarities with ours, in the sense that they also study the latency of AR applications in different setups and investigate the impact of different hardware characteristics at the edge, however it also differs on several points: we consider a type of application that has not been studied in this context (i.e. one that presents an enhanced video stream to the user and not only visual or auditive indications), and present an extensive study about the communication link. In contrast, Gabriel only uses MJPEG over TCP.

Recently, Trinelli et al. [30] investigated Network Function Virtualization for providing computing acceleration for MR by offloading to the edge. The MR application type considered was object detection using machine learning (ML) techniques. In their framework, called NEAR, they consider that the device creating the video stream is different from the one displaying the MR-enhanced stream. In our case, this is done in the same device.

Object recognition using ML is also the focus of Zhang et al. [32]. Their system called Jaguar can recognize an object with low latency, after an offline training phase of the edge part. Contrary to our work, they perform some pre-processing of the video frames on the end device, whereas we do all the processing at the edge.

The field of MR and especially AR is very active, and can be considered as one of the killer app for edge computing [10]. One of the latest realisation platforms is WebAR. In WebAR, a browser is used to provide an AR solution that is supposed to be lightweight and cross-platform. Qiao et al. [22] evaluate the performance of a WebAR implementation that showed lower latency when run at the edge compared to when running on the end device. Different to them, we use an application MR, which is still the most commonly used scenario while WebAR is still in the emerging phase [22].

There are few studies comparing different transmission protocols in client-server relationships in MR-related contexts. For example, Fernandez et al. [12] found that UDP allows twice as many supported clients as TCP in a collaborative augmented reality scenario. However, in this case, the messages exchanged between the clients are about the location, and not about video streams as in our case. When implementing their Gabriel prototype, Ha et al. [14] tested both TCP and UDP and concluded that the difference between them in terms of response time was negligible when transmitting over Wi-Fi. They then decided to use TCP to avoid the complexity added by UDP for maintaining reliable transmission. As our application differs from the ones studied using the Gabriel framework, we compared the two protocols to see if it makes a difference in this case. The NEAR framework of Trinelli et al. [30] only considered TCP.

Finally, among the above prototypes or frameworks implemented by researchers, it is noticeable that only the Gabriel platform is available as open-source software, thus enabling further research and easier comparison of similar works. We believe this is an important aspect and therefore provide our prototype implementation open-source for the community to use.

To summarize, edge computing promises very short latencies but the review above shows that so far no benchmarks with strict timing requirements are used in edge research papers.

3 OFFLOADING MR TO THE EDGE

In this section we first discuss the motivation for performing offloading to the edge, then we define the problem that we are considering in this paper. Finally, we present our developed application, which exploits mixed reality.

3.1 Motivation

MR applications usually require a lot of computation resources for their execution, due to the complexity of the computer vision algorithms and/or graphic algorithms included. This has the consequence that some applications can only be run on powerful machines and not on mobile ones [9], even though those are becoming more and more powerful. This is verified by looking at the MR frameworks released by Google (ARCore) or Apple (ARKit) to run on their smartphones. Both are only supported for a limited range of devices[1,2], which have high CPU power and motion sensors, but can nevertheless get into high temperatures upon this usage.

Even though mobile devices such as smartphones are nowadays quite powerful, developing techniques for offloading at the edge is also an enabler for providing MR on other types of end devices (e.g. wearable devices), that are more resource-constrained than smartphones [26].

Offloading to the cloud, that can be considered as having unlimited resources, is one of the solutions that has been envisioned in the past [15, 17, 21, 25]. However, with the recent advent of edge computing, works such as the one from Chen et al. [10] have shown that using edge is clearly preferable to using cloud for AR applications, because of the reduced latency delay that edge implies.

In addition to latency considerations, offloading to the edge is also preferable from an energy point of view as the telecommunication networks have the highest share of the overall energy consumption of ICT, and the fact that edge computing consists of local traffic is a step in the right direction [4]. Keeping the computation as close as possible to the end users would also keep the energy cost of MR applications as low as possible.

3.2 Problem Description

In this work, we study the process of edge offloading, i.e. moving parts of an application to the edge in order to perform the heavy computation there and free resources in the end devices. The aims are to either enable the application to run on devices with lower capabilities and lower heat, free resources on end devices in order to run other applications, and extend the battery life of end devices running the application. This generic idea of edge offloading is illustrated in Figure 1. The input can for example be sensor data such as a video stream or a GPS location, and the output something that will be displayed to the end users.

We consider the case of an application that gathers input and outputs results in the same end device (as shown in Figure 1). Other

[1]For ARCore: https://developers.google.com/ar/discover/supported-devices
[2]For ARKit: https://www.apple.com/ios/augmented-reality/

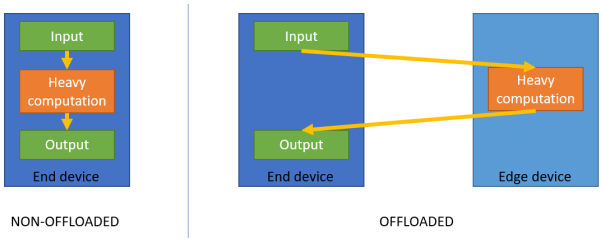


Figure 1: Generic idea of edge offloading

works like the one by Trinelli et al. [30] considered the case where the input and output devices are different.

In order to make an offloaded version of such an application, one needs to first identify the parts of the application pipeline that are resource intensive (in orange on Figure 1) and then to make them execute at the edge instead. Input and output processing on the end device can be more or less resource-heavy depending on the application. In the offloaded scenario, communication is required between the end and the edge devices in order to make the application work. Hence, the quality and characteristics of those two communication links are going to be decisive for the final QoS of the offloaded application.

Ideally, those communication links should:

- Have a **low latency**, in order to keep the final total latency as low as possible
- Have **sufficient bandwidth**, for handling the data required by the application
- Be **reliable**, or have a back-up solution if the link is not available
- Be **secure**, to ensure the application will not be affected by attacks.
- Guarantee **privacy**, as sensitive data may be required for the application

It should be noted that the above problem is described in a generic way and could be used for any application. In this work, we conduct a case study of the above problem in the context of mixed reality, and focus on the first two aspects of the communication links.

3.3 Case Study: Mixed Reality

An MR application aims at enhancing reality with virtual elements. It can be including a few virtual elements, like an Android mascot in a room , or many virtual elements like real persons moving into a virtual city.

The basic phases of an MR application are illustrated in Figure 2. First the reality is captured, usually in the form of a video stream (1), then this stream is analyzed and a virtual representation of the world is constructed (called the *point cloud*) (2). After that, the virtual graphics are created and placed to be at the wanted position (3) and finally the resulting video stream is displayed to the user (4). Next, we describe these in more detail.

Sensor input: The end device can be equipped with different sensors, the most common being a camera that will gather a video stream of the reality. This input can also be combined with other sensory input such as input from the Inertial Measurement Unit, to give more precise information about the end device's movement.

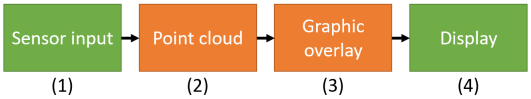


Figure 2: MR application pipeline

Point cloud: In order for the virtual elements to blend nicely in the reality and therefore bring the illusion of reality to the end user, the MR application needs to create a virtual representation of the surroundings of the end devices and to follow the location of the end device in this environment. For example, this will enable the detection of flat surfaces (horizontal or vertical) in order to place virtual elements, such as a 3D virtual touristic guide at a position that makes sense for a human. For example, it would seem strange to have the guide walking on a wall.

In order to create this representation, simultaneous localization and mapping (SLAM) techniques have been developed in the past [8, 13]. Those techniques enable the creation of a 3D map of the surroundings and keep track of the end device's position within it. This 3D map is often modelled as a 3D point cloud. This part is very computationally intensive since advanced computer vision are required for e.g. edge detection, human face detection, and structure from motion, to create the surroundings map.

Graphic overlay: Once the point cloud is created, it will be used to provide the MR functionality, for example to place the virtual element(s) when needed. A visualization of the point cloud can also be output to the user. Depending on how fine the virtual objects should be rendered (only text or 3D objects with use of shadows, refined textures, etc.) the resource requirements for this part vary. After creating the virtual element(s), this part uses the point cloud information to place each element at an appropriate location in the surroundings of the end device, thus creating an overlay to the initial video stream.

Display: Finally the resulting video stream comprising the original video stream with the added virtual graphic overlay is displayed to the end user.

In our case study, we want to offload the parts (2) and (3) of the MR application pipeline shown in Figure 2 as candidates for *heavy computation*. The point cloud creation part (part (2)) is always computationally intensive, whereas the graphic overlay creation (part (3)) may not be. However, it requires the knowledge of (2) in order to work correctly. As the point cloud data is quite important and modified all the time, it is not desirable to transmit it frequently. Parts (1) and (4) are kept in the end device and correspond to the input and the output steps in Figure 1.

4 PROTOTYPE IMPLEMENTATION

In order to make an empirical study of the feasibility of offloading to the edge for an MR application, we developed our own prototype. This is to the best of our knowledge the first available open-source prototype for MR that allows video streaming back to the end device and in-depth study of the communication link.

In the following sections, we present the architecture of the prototype, some design choices and discuss insights gained during the implementation process. The code for the prototype is made fully available online[3,4] in the hope to contribute to further development and studies of MR. In addition to the code, readers interested in implementation details are referred to [16].

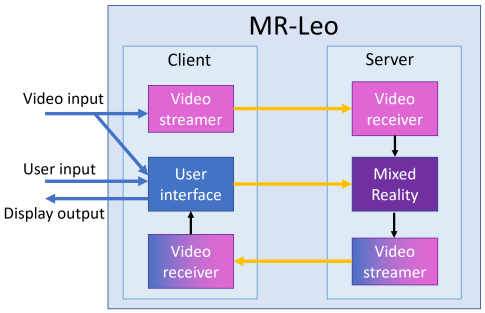


Figure 3: Overview of the MR-Leo architecture.

4.1 Architecture

MR-Leo has a client/server architecture. On the client side (the end device), only the input/output tasks of Figure 1 are performed, corresponding to parts (1) and (4) of the MR pipeline presented in Figure 2. On the server side (the edge device), the heavy computation part of Figure 1 is performed, corresponding to parts (2) and (3) of Figure 2.

To implement the communication link that will transport the required data between the end and the edge devices (yellow arrows in Figure 1), two major aspects had to be dealt with. First, there was a need to perform encoding and decoding of the video stream on both the end and the edge device. Second, a transmission protocol had to be chosen for the video stream. As we want to study different alternatives for those parts, the architecture is made modular to allow switching between different variants for those components.

Figure 3 presents a high-level view of the MR-Leo architecture. The client part takes two types of input: video input (either coming from the end device's camera or a recording) and user input (e.g. pressing the screen to add an MR element). The client outputs via the user interface both the original video stream and the resulting MR-enhanced video stream on the end device display. The video streamer and the video receiver are in charge of preparing the video for transmission, respectively reception over the communication link (including encoding, respectively decoding).

On the server part of MR-Leo, there are two kinds of input possible, both coming from the client part: the encoded video stream and messages indicating user input. The server outputs an MR-enhanced video stream to the client. Mirrored from the client, the arriving video stream has to be decoded and the MR-enhanced stream is encoded before leaving the server. The central part of the server software is the MR component, where the heavy computation described in parts (2) and (3) of Figure 2 is performed.

The MR-Leo architecture is modular, making it easy to test different networking protocols, encoders/decoders or MR frameworks (for example other implementations of SLAM techniques such as CNN-SLAM [29] or LSD-SLAM[11]).

[3] https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_server
[4] https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_client

4.2 Design choices

MR-Leo was implemented to work on an Android smartphone acting as the client and a Linux machine acting as the server. To implement the MR-Leo architecture, we use a mix of already existing software frameworks and libraries and custom-implemented components, when off-the-shelf alternatives were not suitable or available. Figure 3 also indicates the parts we implemented completely ourselves (in blue), the parts where we reused existing open-source libraries (in pink) and the parts for which we modified existing open-source libraries (in purple). For some parts (indicated with a blue/pink fading), it depends on the configuration chosen.

The open-source frameworks that were used were: Gstreamer[5] for implementing the video streaming/receiving parts, ORB-SLAM2 [20] for SLAM techniques for MR, and Pangolin[6] for rendering MR graphics to an image feed. It was out of the scope of this work to evaluate the performance of the SLAM algorithms used. Therefore, we selected the ORB-SLAM2 framework that obtained good results in the SLAM benchmark SLAMBench2 [6], as a representative for state-of-the-art SLAM framework. For the server video streamer and the client video receiver, we use Gstreamer or a custom implementation of an MJPEG encoder/decoder as alternatives in our evaluations below.

4.3 Insights

During the implementation of MR-Leo, we gained several important insights about what to consider when developing an MR application. We describe them here as they might be guidelines for further work in this area.

Currently, the SLAM algorithms used at the edge for creating the point cloud do not have a bounded execution time. This means that it can take variable time analyzing a specific frame. During that time, the next frames cannot be analyzed and are queued in the server. In order to avoid latency accumulation, a resilient mechanism has been implemented to ensure that outdated frames will not be handled by the application and are dropped. However, this should not be happening too often in order to keep the QoS. Therefore, one guideline for developing a responsive MR application is to strive towards handling a frame before the next one comes.

Although the communication between the end and the edge devices and the MR calculations may appear as being two separate parts of offloading to the edge, they are in reality intertwined. Indeed, what is chosen as communication link will have an impact on the quality of the video stream to be used as input for the MR algorithm. If this quality is too low for the chosen MR algorithms, then the QoS of the application will degrade, not because of the choices made for MR but due to the communication link.

Some important benefits of taking off-the-shelf components for implementing MR-Leo are: reduced developing time, access to a community and less frequent (or patchable) bugs. However, this choice also comes with important drawbacks in the context of MR. We noticed that in order to achieve the low latency which was our target for MR at the edge, one has to optimize both the MR framework and create tailored streaming components that are more specialized than the current ones.

[5]https://gstreamer.freedesktop.org/

[6]https://github.com/stevenlovegrove/Pangolin

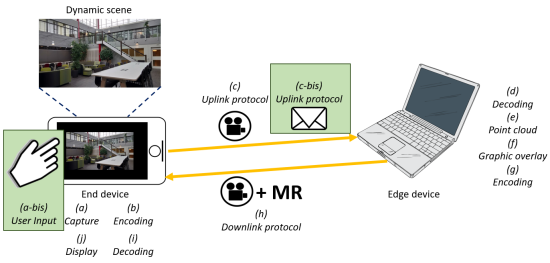


Figure 4: Mixed Reality scenario used in the experiments.

Finally, encoders/decoders used are sensitive to the frame rate of the video stream and irregular frame arrivals will have a negative effect on performance. Therefore, another guideline for developing an MR application is to reduce the difference in time spent for analysis of each frame, so that the MR output is as regular as possible.

5 PERFORMANCE EVALUATION

In this section, we present the experimental setup in which all the following experiments were conducted. Then, we discuss the used metrics and in particular the concept of end-to-end (E2E) latency for mixed reality. After that, we present the evaluation results for our baseline configuration and discuss acceptable performance.

5.1 Evaluation Setup

We first describe the scenario and then the test environment.

5.1.1 Experiment scenario. An experiment consists in capturing a dynamic scene with a smartphone and sending the captured video stream to an edge device that will analyze it to create a point cloud. The scene is dynamic because the camera is moving. Then, the stream is augmented with a visualization of the point cloud sent back to the smartphone. Figure 4 presents this scenario. We can see that in addition to the steps present in Figure 2, the prototype requires additional steps related to the video transmission between the end and the edge devices. In addition, the smartphone user can press a button in order to add a virtual element to the scene. In this case, the steps performed are a bit different, as shown in green on Figure 4. The main difference is that the uplink transmission now only consists of a message indicating the element to be added, and not of the full video stream.

The evaluation in this work consist in comparing different configurations that can be used for the scenario presented in Figure 4. The configurable parts were i) the video compression format used for encoding at the end device (and necessarily, for decoding at the edge device) (i.e. (b) and (d) on Figure 4), ii) the transport protocol used for the uplink transmission ((c)), iii) the video compression format for encoding at the edge device (and necessarily, for decoding at the end device) ((g) and (i)), iv) the transport protocol used for the downlink transmission ((h)). The other parts of the prototype can be modified but this was out of the scope of this work.

Each experiment was conducted 30 times for the same configuration in order to mask any network interference or computing

Figure 5: Test environment.

hardware performance fluctuations. In order to ensure reproducibility between the runs, a video play-back is used instead of the actual camera feed. The test video used is 60 seconds long and is set up in an indoor environment. The full video is available online[7], and has a resolution of 640x480 pixels and a frame rate of 30 fps.

During each experiment run, the addition of a virtual element was automatically triggered five times, every 10 seconds.

5.1.2 Test environment. We first ran the experiments with a first edge/end device pair, hereafter called the *baseline* devices. Then, we conducted a second set of experiences with another edge device or end device that exhibit different characteristics of interest. Those are denoted as the *extra* devices in the paper.

Regarding edge devices, the baseline device is a Lenovo Thinkpad T450s laptop. The laptop runs Ubuntu 18.04 and has 12 GB RAM and an Intel Core i5-5200U CPU (2.2 GHz, 2 cores, 4 threads), which is classified as a high mid-range CPU as of July 2019 [27]. The extra edge device is an HP Elitebook 840 G5. It runs Ubuntu 18.10 and has 16 GB RAM and an Intel Core i7-8550U CPU (1.8 GHz, 4 cores, 8 threads), which is classified as a high end CPU as of July 2019 [28]. Therefore, the extra edge device is expected to be somewhat more powerful than the baseline one.

The baseline end device is an LG G6 smartphone running Android 8.0 and equipped with the Qualcomm Snapdragon 821 mobile platform. It contains a Qualcomm Kryo CPU (2.4 GHz, 4 cores) and 4 GB RAM. The extra end device is a Samsung A5 (2017 model) also running Android 8.0 and equipped with the Samsung Exynos 7880 mobile processor. It contains a Cortex A53 CPU (1.9 GHz, 8 cores) and 3 GB RAM. Compared to the baseline device it is a bit less powerful but provided hardware from a different vendor.

During the experiments, all third-party applications on the end devices were uninstalled, and the internal applications that could be disabled were disabled. On the edge devices, no other application, scheduled tasks or other services were running in the background.

The experiments were performed over a local network set up using an Asus RT-N12 router disconnected from the Internet. The edge device was connected to the network using an Ethernet cable and Gigabit Ethernet (1000 Mbit/sec), and the end device was connected using an 802.11n wireless network. The end and edge devices were placed within one meter from the network gateway, and the same positions were used for all tests using the same devices. Figure 5 shows a picture of the test environment.

[7] https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_video

5.2 Performance Metrics

In order to measure the performance of the prototype application in the different configurations, we study two different aspects. Both are of importance so that the virtual elements integrate seamlessly with the reality, i.e. for QoS. The first aspect is the **latency** of the application. This one should be low in order for the application to be reactive enough the user's movement and interactions. The second aspect is the **throughput** of the application, i.e. how much of the incoming stream is displayed with the added virtual elements to the user. This contributes to QoS by being related to how well the virtual elements integrate with the reality in a seamless manner.

For the latency aspect, the metric commonly used [10] is end-to-end (E2E) latency. However, in the context of this study, this could mean two different things. The first way to consider E2E latency is to measure at the end device the time elapsed between the moment the end user presses the "Add virtual element" button and the moment when the virtual element appears on the display (Steps (a-bis) to (j) on Figure 4). We denote this first latency metric as the *time to virtual element* (T2VE). This is the metric considered for E2E latency by Chen et al. [10].

However, we argue that in our MR scenario, E2E latency could also be understood and measured as the time it takes for a captured video frame to be displayed with the MR enhancement on the display (Steps (a) to (j) on Figure 4). In our scenario, this is expected to be different from the T2VE as the video has to be transmitted to the edge device and back, and not only one way. We denote this second latency metric as the *frame round trip time* (FRTT).

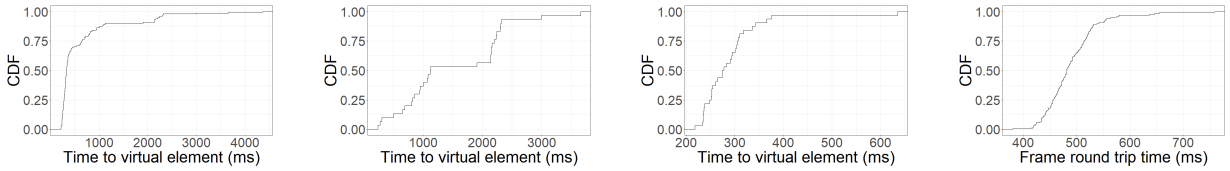
The T2VE is automatically measured by adding in the mixed reality part, in addition to the virtual element, a row of pixels with a uniform color not present in the video and detecting in the end device the appearance of this row. Our tests showed that this detection is lightweight and do not impact the results. Similarly, we modify some frames of the original video to contain a row of pixel with a second color for the FRTT measurements.

The throughput aspect corresponds to how many of the incoming frames are processed by the edge device and how many are discarded due to the resilient mechanism described in Section 4.2. Indeed, in order for the SLAM algorithms to perform well, the number of frames dropped should be as low as possible. We measure this second aspect as the number of frames received at the end device. This way, we account for frames loss in the MR algorithms, but also during transmission, which is part of the prototype under evaluation.

5.3 Evaluation of the Baseline Configuration

In the rest of the paper, the baseline configuration consists of the baseline end device and the baseline edge device using a H.264 video stream transmitted over TCP. Those two were chosen because H.264 is widely used for video streaming, and TCP has been chosen for other MR prototypes [10]. We evaluate this baseline using the performance metrics presented in Section 5.2.

Figure 6a shows the cumulative distribution function (CDF) of the T2VE measurements for the baseline configuration. It can be seen that the value at the 90th percentile is 1122 ms, but also that the it can be as low as 208 ms or as high as 4363 ms. Since this data includes the measurements made during the 30 runs for the five



(a) T2VE, all virtual elements **(b) T2VE, virtual element added at 30 sec (unstable point cloud)** **(c) T2VE, virtual element added at 40 sec (stable point cloud)** **(d) FRTT**

Figure 6: Latency CDFs for the baseline configuration.

times a virtual element is triggered, we performed further analysis to identify what is causing those extreme values.

We find out that in the test video, there is in particular one moment where the SLAM framework has difficulties in constructing the point cloud. This moment happens around 30 seconds, when the camera moves out of the previous scene to film the surroundings. If we isolate the measurements for the virtual element triggered at 30 seconds (presented in Figure 6b), we can see that they show a lot higher latency with the 90th percentile at 2313 ms, more than double compared to the whole dataset. This is a direct consequence of the point cloud being lost in the MR system, so the virtual element cannot be placed and rendered until the point cloud is recovered.

On the contrary, Figure 6c shows the CDF for the T2VE the next time a virtual element is triggered. At this moment in the video, the point cloud creation is stable, and then the T2VE is 6.8 times lower than for the previous one, with a 90th percentile at 342 ms.

During each experiment run, we measure the FRTT for six different frames, with ten seconds within them. Figure 6d shows the CDF for the FRTT measurements. We note two interesting aspects. The first one is that contrary to the T2VE measurements, the FRTT measurements are less spread. This is due to the FRTT measurements occurring when the point cloud is stable. The second aspect is that the FRTT measurements show 1.6 times higher latency at the 90th percentile than the T2VE ones (538 vs 342 ms). This is expected as the "add a virtual element" message is transmitted faster than the video stream itself, in its own data link.

Finally, we find that the baseline throughput varies but in 95% of the cases is superior or equal to 24 fps (not shown due to space limitations).

5.4 Acceptable Performance

In order to evaluate the intrinsic performance of the prototype, we look up for performance guidelines for MR applications in the literature, related to the two aspects of performance investigated.

With regards to the latency aspect, the idea is that the MR enhancement should appear as immediate to the user. In his study of acceptable response time for human-computer interactions, Miller [19] present the limit of 100 ms for the system's answer to be perceived as immediate. 100 ms is also the higher bound that is considered as the limit after which an online action-based game will be perceived as unplayable [24]. However, the latency is dependent on the end device used. For example, mixed reality projected inside VR glasses requires latency to be under 20 ms in order to prevent motion sickness [9]. In our phone-based scenario, we adopt the threshold for acceptable performance as 100 ms.

With regards to the throughput, the displayed video stream should display a high-enough frame rate so that the user cannot identify individual frames. We consider that an acceptable performance on the end device is a video stream returned from the edge device at 24 fps, which is the standard used in the movie industry.

	Comm. service	Point cloud	Graphics	Transmission
Average time spent per frame (ms)	5	31	5	448
Standard deviation (ms)	7.8	6.5	1.7	45.6
% of total FRTT	1%	6%	1%	92%

Table 1: Breakdown of the average FRTT.

5.5 Discussion

With regards to the thresholds defined above, the baseline configuration achieves acceptable performance for the 90th percentile with regards to throughput but not with regards to latency, whether it is T2VE or FRTT. In order to identify which part of the system is the bottleneck, we perform a breakdown of the average FRTT (Table 1).

We can see that the vast majority (92%) of the latency is spent in the transmission. Therefore, we investigate alternatives to the baseline configuration for the video transmission in Section 6. Moreover, the latency breakdown indicates that time is spent in the communication service. In particular, frames are queued before being processed by the edge device due to the MR framework being occupied at processing one of the previous frames. Therefore, it is also relevant to investigate the impact of added computational resources, which we do in Section 7.

Another interesting thing that came out of the baseline configuration evaluation is the fact that the video stream used will have an important impact on the performance of the application. When the point cloud is stable (i.e. measurements at the 20th, 40th and 50th seconds), T2VE is lower than when the SLAM framework used is having issues.

Finally, the difference in measured latency for T2VE and FRTT shows that when studying the performance of an MR application that offloads both the computing and the rendering it is important

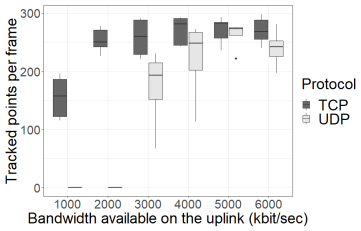
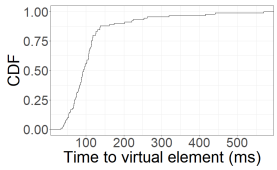
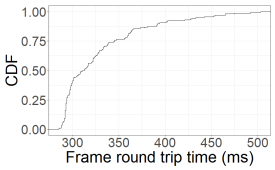


Figure 7: Number of tracked points depending on transport protocol and bandwidth.



(a) T2VE, virtual elements added with a stable point cloud.



(b) FRTT

Figure 8: Latency CDFs for the UDP configuration.

to consider both in order to get a correct idea of how the application is performing.

6 STUDY OF THE COMMUNICATION LINK

The prototype was designed so that it is easy to study the impact of changed modules, e.g. the video compression format. Previous work focused on the computational part. Here, we investigate what could be done in order to decrease the time spent in the communication link, compared to the baseline solution. We study two aspects of the communication link between the end and the edge device: the transport protocol and the video compression format.

6.1 Impact of the Transport Protocol

The transport protocol typically used in related works is TCP. This protocol ensures all the video frames will arrive and in the correct order. However, this comes at the cost of the overhead for the acknowledgment packets. Using UDP, there is no guarantee that all the frames will arrive and in which order, but it is more lightweight. In this section, we study the impact of replacing TCP with UDP in the baseline configuration used in steps (c) and (h) depicted on Figure 4. The rest of the steps are not modified.

We find that changing the transmission protocol has an impact on three aspects of the communication link: the minimum bandwidth required, the latency of the MR application (both T2VE and FRTT), and the application throughput.

As described before, the MR framework requires images of a high enough quality in order to create a point cloud. If the video frames have been compressed so much that interesting features are not distinguishable anymore, then the MR application will not be able to deliver its service. In the prototype, how much the frames from the video stream are compressed is directly related to how much bandwidth is available on the link. Therefore, we performed a separate experiment that step-wise increased the bandwidth available on the communication link and measured the number of feature points tracked by the MR framework. The results ae shown in Figure 7 and we found that a suitable and stable amount of feature points (around 250) can be achieved for a bandwidth of 2000 kbit/sec for TCP and 4000 kbit/sec for UDP. The difference is due to the fact that UDP has to transmit complete frames all the time because some could be lost whereas TCP only needs to send the difference between frames most of the time. We use those bandwidth values for the latency measurement in order to compare latency and throughput for a similar frame quality used as input to the communication service.

With regards to latency, Figure 8a shows the CDF of T2VE measurement for the UDP configuration, with data when the point cloud is stable. At the 90th percentile, the T2VE with UDP is 3.7 times lower than for the baseline using TCP (177 ms vs 654 ms).

Similarly, Figure 8b shows the CDF for FRTT with UDP. We observe that the behaviour is similar to the one for TCP but the latency is also lower for this metric, with a 90th percentile at 392 ms. However, the gap between TCP and UDP is lower for FRTT; UDP is only 1.4 times faster than TCP for this metric.

With regards to throughput, the acceptable frame rate of 24 fps is achieved 79% of the time with UDP, which is lower than for TCP. This is explained by the fact that frame losses are going to be higher with UDP since the protocol has no guarantee for receiving them and this can impact not only the frames loss during the communication but also the performance of the SLAM framework as it may be harder to maintain the point cloud if frames are missing (meaning that the computing time per frame at the edge may increase).

6.2 Impact of the Video Compression Format

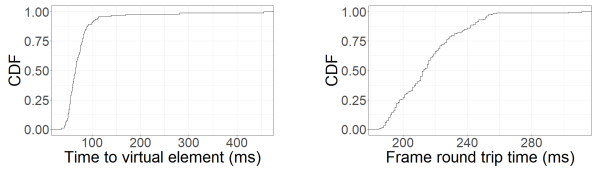
Another interesting part of the communication link is the video compression format, because it will have an effect on how long the encoding/decoding phases will be. As an alternative to the common H.264 format used in the baseline, we use in this section another common format: the MJPEG one, for the steps (g) and (i) of Figure 4. No other step is modified and the bandwidth used is the same.

We find that using the MJPEG format improves the latency (both T2VE and FRTT), as shown in Figures 9a and 9b. The latency at the 90th percentile is indeed 6.7 times lower (98 ms) for T2VE and 2.2 times lower (245 ms) for FRTT.

With regards to the throughput, the MJPEG configuration performs similarly to the H.264 one, with a frame rate superior or equal to 24 fps 96% of the time. Since the MR edge part gets exactly the same input as the baseline, this shows that the MJPEG encoding/decoding does not impact the frame rate negatively at the same time as it improves the latency.

6.3 Summary

We showed that using UDP instead of TCP has the potential of reducing the latency of the MR application. However, this comes at the cost of higher bandwidth required for the communication link and the MR framework has to tolerate potential loss of frames. Thus, application developers or edge infrastructure providers will have to keep these trade-offs in mind when deploying their services.



(a) T2VE, virtual elements added with a stable point cloud. **(b) FRTT**

Figure 9: Latency CDFs for the MJPEG configuration.

Our study of the video compression format showed that our custom implementation of an MJPEG encoder/decoder for the downlink keeps the high frame rate of GStreamer using H.264 but at a lot lower latency. Further quality of experience (QoE) studies are however required in order to determine the impact of using a different video compression format on the quality of the resulting video depending on the bandwidth used. Indeed, the current prototype is not affected with regards to QoS by an unsufficient bandwidth on the downlink, but the end user QoE might be.

One further performance gain is to use hardware encoding instead of the software encoding used at the moment. Preliminary tests performed indicate that this has the potential of lowering the latency even more when using UDP.

7 A DEEPER LOOK AT FRTT

FRTT is important in the studied MR application as it has a high impact on the QoS. If the FRTT is too high, the user will notice that the video stream containing the virtual elements is not matching what he/she is currently filming.

In this section we test two hypotheses. As visible in Table 1, although the bulk of the FRTT is spent for transmission, around 1% of the total (corresponding to 5% of the acceptable performance) is spent in queuing frames in the edge part since the MR framework is already busy. Therefore, the first hypothesis (H1) is that using a more powerful edge device will remove this queuing time, and thus reduce the FRTT. Then, as all the heavy computations are performed at the edge, the FRTT should not be impacted by changing the end device. This is the second hypothesis (H2).

7.1 H1: A better CPU improves FRTT

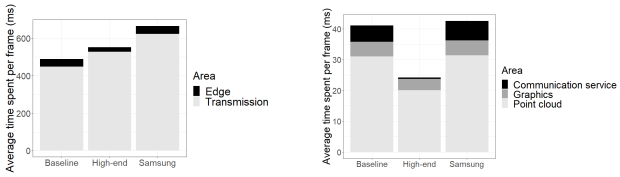
In order to test H1, we compare the performance of the prototype application when running on the baseline edge device and on the extra, high-end edge device. Table 2 summarizes measurement statistics, where MR includes point cloud creation and graphics.

In particular, we present in Figure 10a a breakdown of the average FRTT for different configurations. The results are surprising, because the average FRTT for the high-end configuration is 1.13 times higher than for the baseline configuration. Hence, an edge device with a more powerful CPU does not necessarily reduce the FRTT, it can actually increase it.

Even though the overall FRTT is increased, we zoomed in on the part of it that is spent at the edge. Figure 10b shows that the queuing time in the communication service is indeed reduced when using the high-end configuration. The MR framework is also executing

		Avg	σ	90th	Min	Max
Baseline	Graphics	5	2	6	3	50
	MR	36	7	44	19	131
	Edge	41	12	58	20	132
	FRTT	489	50	538	380	759
High-end	Graphics	4	1	4	2	18
	MR	24	4	27	15	93
	Edge	24	5	27	16	93
	FRTT	552	50	635	472	676
Samsung	Graphics	5	2	6	3	49
	MR	36	7	45	20	133
	Edge	42	13	61	20	138
	FRTT	666	37	705	591	827

Table 2: Performance measurements statistics (ms).



(a) Average FRTT **(b) Average edge processing time**

Figure 10: Latency breakdown for different configurations.

faster. However, the high-end configuration had negligible impact on the time spent on graphics. We conjecture that to decrease this part, other improvements such as using graphic accelerators are needed, if the SLAM framework used can take advantage of it.

7.2 H2: FRTT is smartphone-independent

Chen et al. [10] showed that in the context of wearable devices, there is a performance impact based on the hardware used. With major computations done at the edge, we now compare the baseline device with an additional smartphone (Samsung), that, although a bit less powerful than the baseline one, should be fully capable of handling the MR application. The results are presented in Table 2.

Unexpectedly again, the values show that the Samsung configuration as 1.36 times higher FRTT on average. This indicates that some part of the application is handled differently by the two end devices considered. By looking at Figures 10a and 10b, it appears that it is the transmission part that behaves differently, i.e. either the encoding/decoding, the WiFi transmission or the way the TCP protocol is handled. Interestingly, preliminary results with UDP showed that the difference in FRTT in this case is lower between the two configurations, indicating that the protocol handling should be investigated further.

8 CONCLUSION

Mixed reality applications are a good candidate for offloading at the edge. Indeed, they require heavy computation and very low latency in order to deliver high quality of service to the end user. However, deploying an MR application that performs video streaming towards

the edge and back is not trivial due to the different components that need to be brought together.

In this work, we present an open-source prototype for such an MR application and dissect its performance with regards to latency and throughput using different alternatives for the communication link and the hardware used. We found that bringing down the time spent in the communication link is critical for such MR applications and we see the different alternatives considered in this paper as a first step towards further studies of this aspect.

Future works include investigating other possibilities for reducing the latency of the MR application such as using hardware acceleration for encoding/decoding, using other SLAM frameworks, using other types of protocols and dissecting combinations of those. Moreover, it would be interesting to study the application with regards to its energy usage. Finally, this work could be extended to execute within a future 5G deployment where other protocols are in use and where hardware accelerators are exploited. The MR prototype is provided open-source to encourage other researchers to experiment with it and test other interesting aspects of MR offloading at the edge.

ACKNOWLEDGMENTS

This work was supported by the Swedish National Graduate School in Computer Science (CUGS). The authors would like to thank Marcus Gårdman and his team at Ericsson for our discussions that led to this work.

REFERENCES

[1] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2018. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal* 5, 1 (2018), 450–465. https://doi.org/10.1109/JIOT.2017.2750180

[2] Surin Ahn, Maria Gorlatova, Parinaz Naghizadeh, and Mung Chiang. 2019. Personalized Augmented Reality via Fog-based Imitation Learning. In *Proceedings of the Workshop on Fog Computing and the IoT (IoT-Fog '19)*. 11–15. https://doi.org/10.1145/3313150.3313219

[3] Surin Ahn, Maria Gorlatova, Parinaz Naghizadeh, Mung Chiang, and Prateek Mittal. 2018. Adaptive Fog-Based Output Security for Augmented Reality. In *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network '18)*. 1–6. https://doi.org/10.1145/3229625.3229626

[4] Ehsan Ahvar, Anne-Cecile Orgerie, and Adrien Lébre. 2019. Estimating Energy Consumption of Cloud, Fog and Edge Computing Infrastructures. *IEEE Transactions on Sustainable Computing* (March 2019), 1–1. https://doi.org/10.1109/TSUSC.2019.2905900

[5] Mafkereseb K. Bekele, Roberto Pierdicca, Emanuele Frontoni, Eva Savina Malinverni, and James Gain. 2018. A Survey of Augmented, Virtual, and Mixed Reality for Cultural Heritage. *J. Comput. Cult. Herit.* 11, 2, Article 7 (March 2018), 36 pages. https://doi.org/10.1145/3145534

[6] Bruno Bodin, Harry Wagstaff, Sajad Saecdi, Luigi Nardi, Emanuele Vespa, John Mawer, Andy Nisbet, Mikel Luján, Steve Furber, Andrew J. Davison, Paul H. J. Kelly, and Michael F. P. O'Boyle. 2018. SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 1–8. https://doi.org/10.1109/ICRA.2018.8460558

[7] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. ACM, New York, NY, USA, 13–16. https://doi.org/10.1145/2342509.2342513

[8] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. 2017. Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving. *IEEE Transactions on Intelligent Vehicles* 2, 3 (Sep. 2017), 194–220. https://doi.org/10.1109/TIV.2017.2749181

[9] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. 2017. Mobile Augmented Reality Survey: From Where We Are to Where We Go. *IEEE Access* 5 (2017), 6917–6950. https://doi.org/10.1109/ACCESS.2017.2698164

[10] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, Daniel Siewiorek, and Mahadev Satyanarayanan. 2017. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17)*. 14:1–14:14. https://doi.org/10.1145/3132211.3134458

[11] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Computer Vision – ECCV 2014*. 834–849. https://doi.org/10.1007/978-3-319-10605-2_54

[12] Víctor Fernández, Juan Manuel Orduña, and Pedro Morillo. 2014. Server implementations for improving the performance of CAR systems based on mobile phones. *Journal of Network and Computer Applications* 44 (2014), 72 – 82. https://doi.org/10.1016/j.jnca.2014.04.012

[13] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. 2015. Visual Simultaneous Localization and Mapping: A Survey. *Artif. Intell. Rev.* 43, 1 (Jan. 2015), 55–81. https://doi.org/10.1007/s10462-012-9365-8

[14] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14)*. 68–81. https://doi.org/10.1145/2594368.2594383

[15] Zhanpeng Huang, Weikai Li, Pan Hui, and Christoph Peylo. 2014. CloudRidAR: A Cloud-based Architecture for Mobile Augmented Reality. In *Proceedings of the 2014 Workshop on Mobile Augmented Reality and Robotic Technology-based Systems (MARS '14)*. 29–34. https://doi.org/10.1145/2609829.2609832

[16] Johan Lindqvist. 2019. *Edge Computing for Mixed Reality*. Master's thesis. Linköping University.

[17] Xun Luo. 2009. From Augmented Reality to Augmented Computing: A Look at Cloud-Mobile Convergence. In *2009 International Symposium on Ubiquitous Virtual Reality*. 29–32. https://doi.org/10.1109/ISUVR.2009.13

[18] Paul Milgram and Fumio Kishino. 1994. A taxonomy of mixed reality visual displays. *IEICE Trans. Inf. Syst* 77, 12 (1994), 1321–1329.

[19] Robert B. Miller. 1968. Response Time in Man-computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS '68 (Fall, part I))*. 267–277. https://doi.org/10.1145/1476589.1476628

[20] Raúl Mur-Artal and Juan D. Tardós. 2017. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (Oct 2017), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

[21] Nayyab Zia Naqvi, Karel Moens, Arun Ramakrishnan, Davy Preuveneers, Danny Hughes, and Yolande Berbers. 2015. To Cloud or Not to Cloud: A Context-aware Deployment Perspective of Augmented Reality Mobile Applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. ACM, New York, NY, USA, 555–562. https://doi.org/10.1145/2695664.2695880

[22] Xiuquan Qiao, Pei Ren, Schahram Dustdar, and Junliang Chen. 2018. A New Era for Web AR with Mobile Edge Computing. *IEEE Internet Computing* 22, 4 (2018), 46–55. https://doi.org/10.1109/MIC.2018.043051464

[23] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39. https://doi.org/10.1109/MC.2017.9

[24] Ryan Shea, Jiangchuan Liu, Edith C.-H. Ngai, and Yong Cui. 2013. Cloud gaming: architecture and performance. *IEEE Network* 27, 4 (July 2013), 16–21. https://doi.org/10.1109/MNET.2013.6574660

[25] Ryan Shea, Andy Sun, Silvery Fu, and Jiangchuan Liu. 2017. Towards Fully Offloaded Cloud-based AR: Design, Implementation and Experience. In *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys'17)*. 321–330. https://doi.org/10.1145/3083187.3084012

[26] Bowen Shi, Ji Yang, Zhanpeng Huang, and Pan Hui. 2015. Offloading Guidelines for Augmented Reality Applications on Wearable Devices. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM '15)*. 1271–1274. https://doi.org/10.1145/2733373.2806402

[27] PassMark Software. 2019. PassMark CPU Benchmarks - High Mid Range CPUs. https://www.cpubenchmark.net/mid_range_cpus.html Accessed 19th July 2019.

[28] PassMark Software. 2019. PassMark Intel vs AMD CPU Benchmarks - High End. https://www.cpubenchmark.net/high_end_cpus.html Accessed 19th July 2019.

[29] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. 2017. CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6565–6574. https://doi.org/10.1109/CVPR.2017.695

[30] Marco Trinelli, Massimo Gallo, Myriana Rifai, and Fabio Pianese. 2019. Transparent AR Processing Acceleration at the Edge. In *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '19)*. 30–35. https://doi.org/10.1145/3301418.3313942

[31] Lei Zhang, Andy Sun, Ryan Shea, Jiangchuan Liu, and Miao Zhang. 2019. Rendering Multi-party Mobile Augmented Reality from Edge. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19)*. 67–72. https://doi.org/10.1145/3304112.3325612

[32] Wenxiao Zhang, Bo Han, and Pan Hui. 2018. Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking. In *Proceedings of the 26th ACM International Conference on Multimedia (MM '18)*. 355–363. https://doi.org/10.1145/3240508.3240561