**Noname manuscript No.**
(will be inserted by the editor)

# Explora-VR: Content Prefetching for Tile-based Immersive Video Streaming Applications

Leandro Ordonez-Ante* · Jeroen van der Hooft ·
Tim Wauters · Gregory Van Seghbroeck ·
Bruno Volckaert · Filip De Turck

Received: date / Accepted: date

**Abstract** Despite the growing popularity of immersive video applications during the last few years, the stringent low latency requirements of this kind of services remain a major challenge for the existing network infrastructure. Edge-assisted solutions compensate for network latency by relying on cache-enabled edge servers to bring frequently accessed video content closer to the client. However, these approaches often require historical request traces from previous watching sessions or adopt passive caching strategies subject to the *cold-start* problem and prone to playout freezes. This paper introduces Explora-VR, a novel edge-assisted content prefetching method for tile-based 360° video streaming. This method leverages the client's rate adaptation heuristic to preemptively retrieve the content that the viewer will most likely watch in the upcoming segments, and loads it into a nearby edge server. At the same time, Explora-VR incrementally builds a dynamic collective buffer for serving the requests from active streaming sessions based on the estimated popularity of video tiles per segment. An evaluation of the proposed method was conducted on head movement traces collected from 48 unique users while watching three different 360° videos. Results show that Explora-VR is able to serve over 98% of the client requests from the cache-enabled edge server, leading to an average increase of 2.5× and 1.4× in the client's perceived throughput, compared to a conventional client-server setup and a *least recently used* caching policy, respectively. This enables Explora-VR to serve higher quality

L. Ordonez-Ante (✉) · J. van der Hooft · T. Wauters · G. Van Seghbroeck · B. Volckaert · F. De Turck
Department of Information Technology, Ghent University - imec, IDLab
Technologiepark Zwijnaarde 126
B-9052 Gent, Belgium
Tel.: +32 9 33 14940
Corresponding Author E-mail: Leandro.OrdonezAnte@UGent.be
E-mail: {Jeroen.vanderHooft, Tim.Wauters, Gregory.VanSeghbroeck, Bruno.Volckaert, Filip.DeTurck}@UGent.be

video content while providing a freeze-free playback experience and effectively reducing network traffic to the content server.

**Keywords** Virtual reality · Video streaming · Content prefetching · Collective buffering · Edge-assisted streaming

## 1 Introduction

The recent outbreak of the COVID-19 pandemic has forced a radical shift in reality for a vast majority of the human population. Given the strict restrictions on mobility and social contact, people were compelled to move several aspects of their daily life into the digital world. These circumstances have boosted the interest in 360° immersive video applications (*augmented and virtual reality—AR/VR*) as a means to provide realistic and engaging user experiences, that make up for the lack of presence and physical interaction [1,2,3]. However, the stringent demands in terms of bandwidth and very low latency of AR/VR applications still represent a major challenge for the existing network infrastructure [4].

For services relying on VR headsets for content delivery, the delay perceived by the user is a critical factor for determining the overall experience. Research on this topic signals that the motion-to-photon (MTP) latency for VR displays should be less than 20 ms to prevent the perception of scene instability and *cybersickness* [5,6]. For on-demand tile-based 360° video streaming in particular, many of the existing studies have focused on mitigating the effect of latency by increasing viewport prediction accuracy and applying *HTTP adaptive streaming* (HAS) methods to adapt the quality of the requested content to the network conditions [7,8]. While these approaches achieve a rational use of the bandwidth as perceived at the client's side, the network latency due to distant content servers can still substantially degrade the viewer experience.

As an answer to this problem, network-supported solutions leveraging cache-enabled edge servers have been proposed [9,8]. The idea behind these approaches consists of bringing frequently accessed video tiles closer to the client; this offsets the network delay, which in turn leads to a significant improvement in the quality of the delivered content. This is, however, easier said than done: the high variety of possible viewport configurations—due to the freedom of device orientation, added to different network conditions—makes it hard to determine a priori the set of tiles that should be cached. In this sense, network-supported solutions often rely on log traces obtained from previous streaming sessions to estimate the popularity of the content, and/or adopt passive caching strategies in which only those tiles that are requested get cached at the edge server. These approaches entail two fundamental problems: (*i*) historical request traces are not always available for every piece of content, and (*ii*) the *cold-start* problem: early users would barely experience any improvement from having a cache nearby, due to the fact that most of their requests for content end up being forwarded to the origin server.

To address these issues, in this paper we introduce Explora-VR, a content prefetching mechanism for tile-based immersive video streaming. Our solution introduces two fundamental changes in the traditional workflow of content consumption for this kind of services: (*1*) the early advertising of the outcome of the viewport prediction and rate adaptation algorithm, running on the *head-mounted display* (HMD), and (*2*) the incremental building of a collective buffer that incorporates fixation patterns shared by the viewers. The rationale behind this is two-fold:

1. The information on the predicted user's viewport is forwarded to the cache-enabled edge server before the client's device starts buffering content. The edge server uses this information to preemptively retrieve—at a given quality level—the video tiles that the user is likely to watch in the upcoming segments, and it loads them into memory. Then the client starts consuming the video content from a closer server. In these circumstances, a HAS client would perceive that the content is downloaded with low latency, leading to a high network throughput estimation, and consequently to an increase in quality of the requested video tiles.
2. Since having multiple clients consume the same VR content within a small time window is a common use case (e.g., the *on-demand*, *near-live* scenario when a content provider premieres a new release), we have devised a stream-processing pipeline which enables combining the different user predicted viewports into a dynamic collective buffer (henceforth referred to as *DCoB*) which is built and refined incrementally as new users join the streaming session. The purpose of this DCoB is to serve as a cache holding frequently accessed content, preventing the edge server from flooding the content server with duplicate requests.

This paper presents the following three main contributions of the solution we propose: (*1*) an edge-assisted, content-agnostic mechanism that proactively downloads the video tiles that an individual viewer is likely to watch in the near future (*2*) the formal definition of the data structure and stream processing pipeline behind the DCoB, which enables low-latency delivery of 360° video content to multiple users taking part of an on-demand, near-live streaming scenario, and (*3*) the experimental evaluation of the proposed approach on a public dataset which comprises the viewport traces from 48 users, collected throughout immersive video sessions. We have benchmarked the Explora-VR prefetching mechanism against a conventional client-server configuration (without caching/prefetching), and a setup implementing a traditional *least-recently used* (LRU) caching replacement policy. Results show that the devised prefetching mechanism substantially improves the quality of experience (QoE) perceived by the viewer, in terms of video quality, startup latency, and occurrence of playout freezes, while reducing the backhaul traffic and content server's load.

It should be pointed out that it is not in the scope of this paper to reach an optimal trade-off between network resource consumption and video delivered quality, as is the case for approaches in the literature such as [10] and [11].

Our work is focused on investigating data processing methods for enabling preemptive retrieval of immersive video content which are able to adapt to the fixation patterns of multiple concurrent viewers. To achieve this we leverage the computing resources of cache-enabled edge nodes, and we rely on existing methods for client-side viewport prediction and tile-based rate adaptation such as those introduced in [12] and [13].

The remainder of this paper is structured as follows. Section 2 discusses the related work. Section 3 describes the detailed description of the techniques behind the content prefetching mechanism for tile-based 360° video streaming. Section 4 elaborates on the architecture of a proof-of-concept implementation of the proposed approach. Section 5 presents the experimental setup and the results derived from the evaluation. Finally, conclusions and perspectives for further research are provided in Section 6. Table 1 below provides the list of acronyms used throughout the paper.

Table 1: List of acronyms used in this paper

| **Acronym** | **Description** |
|---|---|
| CDN | Content Delivery Network |
| CRF | Constant Rate Factor |
| CTF | Center Tile First |
| DCoB | Dynamic Collective Buffer |
| DNN | Deep Neural Networks |
| DRL | Deep Reinforcement Learning |
| FoV | Field of View |
| FPS | Frames per second |
| GOP | Group of Pictures |
| HAS | HTTP Adaptive Streaming |
| HEVC | High Efficiency Video Coding |
| HMD | Head-Mounted Device |
| LRU | Least-Recently Used |
| QoE | Quality of Experience |
| RTT | Round Trip Time |
| VR | Virtual Reality |

## 2 Related Work

Immersive video applications are typically bandwidth-hungry and highly sensitive to latency. A large body of research in this field has been devoted to develop efficient mechanisms of content delivery. Existing approaches can be grouped into three categories according to the main focus of their respective contribution, namely *client-driven*, *server optimization*, and *edge-assisted* solutions.

### 2.1 Client-driven HAS streaming for tile-based 360° video

To improve transmission efficiency, approaches in this category divide an equirectangular projection of the spherical video into several rectangular areas of the same size, referred to as tiles. By implementing said tiling scheme, the client can opt to prioritize the tiles that overlap the viewer's viewport and request them in a higher quality representation than the tiles that are not visible to the user. Representative of these tile-based viewport-dependent adaptive video streaming solutions include the approaches by Hosseini, M. [14], Xie et al. [15], Graf et al. [16], Nguyen et al. [17], and van der Hooft et al. [13]. These works are fundamentally focused on addressing two main challenges: (*i*) *viewport prediction*: anticipate user movements to ensure content is timely displayed following the *field of view* (FoV) of the user; and (*ii*) quality of experience (QoE): providing a smooth, responsive watching experience at the highest possible video quality that the best-effort network can deliver [8]. In essence, these approaches adopt traditional HTTP adaptive streaming techniques, and augment them to support tile-based content delivery, while meeting the stringent demands in terms of latency and interactivity of omnidirectional video streaming. Although these solutions allow for an efficient use of the link capacity, they are still highly sensitive to network latency due to content servers situated in distant locations which severely degrades the user experience.

### 2.2 Server optimization solutions

This category comprises works mainly focused on maximizing viewer's quality of experience while optimally allocating server and network resources. Long et al. [10] propose a solution to the problem of optimal transmission resource allocation on the server-side given a specific requirement of video quality from the viewer, as well as the optimal encoding rate for each video tile given a certain transmission energy budget. The solution contemplates exploiting several multicast opportunities that involve balancing trade-offs between video quality, computation, and consumption of communication resources. One of the implications of the proposed multicasting mechanisms is that the server might transmit video tiles at a higher quality representation than that requested by a certain client. In such a case, the client application would incur a processing cost in order to scale down the received video tile to the appropriate quality representation. Building upon [10], the work by Zhao et al. [11] investigates the impact of viewport prediction on adaptive streaming of tiled 360° video in a multi-carrier wireless system. The authors consider a setup with a multi-antenna base station from which video content is transmitted to one or multiple single-antenna clients. Within the scope of said setup, authors propose a framework that optimizes the downlink subcarrier allocations as well as the encoding rates for tiles and FoVs at the server-side. The solution proposed in [11] aims at maximizing the video quality delivered to the clients while controlling the rebuffering time for different levels of certainty about the outcome of the viewport prediction. It is noteworthy that the optimization investigated in [11] rely on methods that operate the radio link layer, which are out of the scope of the work we present in this paper.

Another approach that fits within this category is introduced by Shi et al. in [5]. The authors of this approach propose a remote rendering solution in which the server is able to stream only the scenes within the user's FoV plus a margin area around it whose width depends on the perceived system latency. Instead of a tiling scheme, the server uses an adaptive cropping filter that adjusts the delivered content to the fraction of the VR video overlapping with the current user viewport. A design decision made by the authors of this approach consist in minimizing the use of video buffering to reduce the system's response latency. In consequence, the proposed remote rendering solution is sensitive to network jitters and prone to frame dropping. Furthermore, the authors do not provide a clear indication concerning the performance of the proposed solution under high server load (i.e., serving multiple concurrent viewers).

### 2.3 Edge-assisted solutions

Thanks to the recent availability of public datasets on Virtual Reality (VR) video streaming —such as those by Lo et al. [18], David et al. [19], Fremerey et al. [20], and Wu et al. [21], among others— there has been an increasing interest in investigating methods for mining behavioral patterns from user movement traces. According to the study by Rossy et al. [22], navigation trajectories followed by viewers with high affinity exhibit patterns that can be used for optimizing the content delivery in streaming systems. Approaches aligned with this idea are often labelled as *edge-assisted* or *network-supported* solutions. Papaioannou et al. [23] addressed the problem of optimal caching for tile-based VR video streaming in the wireless edge network. Specifically, the solution introduced in [23] formulates a tile and tile resolution caching policy that aims at minimizing the error between the cached and requested content. The authors studied a static caching scenario in which the caching decision is made upfront, based on statistical data of the tile resolution demands from past watching sessions. Similarly, Mahzari et al. [24] explored the application of edge caching as a measure to compensate for network latency, and offload the content servers and backhaul network. The authors of this work conceived a FoV-aware caching policy based on a bayesian model which takes in the sequence of requests made by previous viewers. The proposed model gauges the popularity of individual tiles, and makes decisions on which content to cache/evict based on said metric. Similarly, Maniotis and Thomos [25] devised a cache replacement strategy for tile-based omnidirectional video, supported by a *deep reinforcement learning* (DRL) framework. This strategy takes into account the popularity of both videos and individual tiles. The authors introduced the concept of *virtual viewports* defined as the most popular video tiles resulting from the overlapping FoV of multiple users. To learn the optimal policy for tile placement in the cache, the DRL framework first requires to train a deep neural network (DNN) on past user requests.

These approaches (and related proposals such as [26, 27, 28, 29]) have proven the pertinence and substantial benefit of edge caching to improve QoE in

360° video services, while reducing the load on the core network. However, these solutions often require an *offline stage* in which they fit a certain data model to traces of user requests. Afterward, in a subsequent *online stage*, this model is used to make decisions on which content to cache/evict, according to the demands from new users consuming the streaming service. In addition, the studies discussed above adopt a passive approach to caching, i.e. tiles are stored into the edge-server memory only after they have been requested. Under these circumstances, early viewers would experience little benefit from the caching strategy in place, an issue that is commonly referred to in the literature as the *cold-start* problem [30]. Using a cold cache translates into cache miss events, which in turn increases the likelihood of playback freezes, since user requests have to be relayed back to the content server, thus incurring additional latency. To counter this issue, we propose a new FoV-aware content prefetching approach for tile-based adaptive 360° video streaming. This approach takes advantage of existing viewport prediction techniques to preemptively retrieve and cache the video content that the viewer is most likely to consume in the upcoming segments. Additionally, this mechanism does not rely on training data from historical traces as it is able to learn a *collective viewport* on the fly, out of the requests made by viewers with active streaming sessions. The content inside the collective viewport dynamically adapts in response to the content that is most demanded by the audience at a given point in time, which makes this approach specially appealing for near-live immersive video streaming applications.

## 3 Explora-VR: Approach Overview

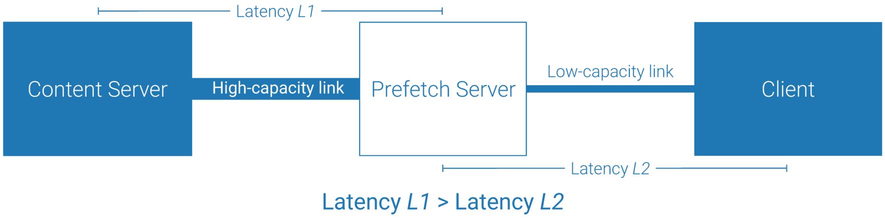


Fig. 1: High-level component view of the VR content prefetching scenario. The link between *Content* and *Prefetch* servers features a larger capacity and higher latency than the one between *Prefetch* server and *Client*.

Figure 1 illustrates the components that make up the content prefetching mechanism we propose. This mechanism is deployed on a cache-enabled edge server acting as a transparent proxy between the client and the content server. In this section we elaborate on the techniques that lay the foundation of our solution, namely (*1*) the early advertising of the outcome of the viewport prediction and rate adaptation algorithm, and (*2*) the *dynamic collective buffer* (DCoB).

### 3.1 Viewport Prediction Advertising and Prefetching

In immersive video applications based on 360° video, it is common for content to be segmented not only in time but also in the spatial dimension. The HEVC/H.265 standard, for instance, allows to split an equirectangular projection of the content into $m \times n$ tiles of the same resolution. By adding this spatial dimension, clients can prioritize the content within the user's *field of view*, assigning a higher quality to specific regions of the video, hence making more optimal use of the bandwidth resources [31,32].

To prevent buffer starvation and ensure a smooth playback in these highly interactive applications, traditional HAS methods need to be augmented. HAS clients for VR applications rely on techniques for predicting the users' target field of view or viewport, and rate adaptation heuristics to fine-tune the quality level of the requested content in response to the users' movements and network conditions [7,32,16].

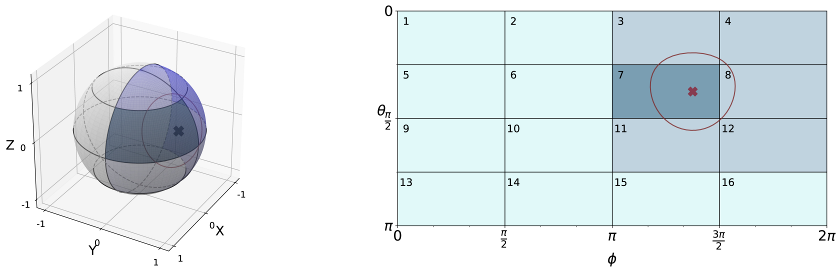
Several methods have been introduced for viewport prediction in tile-based VR video streaming over the last years. On the one hand, *content-agnostic* approaches estimate the trajectory the viewer is likely to follow based on the viewport center locations of the last few milliseconds. To do so, some of these approaches use linear projection on the previous viewport positions [31,33, 34,13], while others rely on machine learning models trained on user movement traces [35,36]. *Content-aware* techniques on the other hand, attempt to anticipate user movements based not only on an estimation of the viewer's trajectory, but also on specific features derived from the video content itself such as *image saliency*, *fixation density* and *object motion maps* [37,38,39,40].

In this work, we adopt the content-agnostic method proposed by van der Hooft et al. in [12] for predicting the user's viewport. In contrast to other content-agnostic solutions that assume the user moving on a path in the two-dimensional space defined by the equirectangular projection of the video, the method proposed in [12] models the viewer's movement as a trajectory on the *unit sphere*'s surface. In this way, the future location of the viewport center is estimated by unidirectionally extending the path covered by the viewer thus far across the surface of the unit sphere (*spherical walk*). This approach to viewport prediction provide a more natural approximation of the viewer's motion within the 360° video scene. This allows for a more accurate prediction compared to alternative content-agnostic solutions using linear extrapolation of the user's trajectory over the equirectangular projection of the video.

It is worth noting that the content prefetching mechanism we propose does not involve any substantial modification to the adopted viewport prediction scheme. Besides, while we favor the use of spherical-walk based viewport prediction—mainly due to its enhanced accuracy—the devised prefetching mechanism is easily compatible with other alternative content-agnostic viewport prediction methods such as those proposed by Petrangeli et al. [31] and Xu et al. [34].

Along with the viewport prediction scheme based on spherical walks, we also adopt the *Center Tile First* (CTF) rate adaptation heuristic proposed in [13]. The intent behind this heuristic is to maximize the quality level for the video tiles located closer to the viewport center. In doing so, tiles from an equirectangular VR video are ranked according to the great-circle distance between their center and the viewport predicted location. The closer a certain tile is to the viewport center, the higher its priority and the quality representation that gets assigned to it.

As illustration, consider the example in Figure 2 for a 4 × 4 tiling scheme and two quality levels. The diagram outlines both the viewport (circular area on the sphere in Figure 2a) and viewport center (indicated as a cross mark). In this example, the CTF heuristic has prioritized the six tiles that lie closer to the viewport center, assigning them a high quality representation.



(a) Viewport projection in the unit sphere. Based on [32]

(b) Equivalent equirectangular projection of the viewport. Based on [32]

Fig. 2: Example of the application of the CTF rate adaptation heuristic for a setup with a 4 × 4 tiling scheme and two quality levels: The highest quality representation gets assigned to the blue-shaded tiles, while the remaining ones are requested in the lowest quality. The number of high/low quality tiles in this example is arbitrary as it depends on the network conditions between client and server.

The output of the rate adaptation heuristic is represented as an array that encodes the tile ranking, along with the quality level assigned to each of the tiles. Traditionally, a VR client would take said array and download each of the tiles, at the specified quality level, into the playout buffer. The prefetching mechanism we propose contemplates an extra step: forwarding the rate adaptation result to the cache-enabled edge server as soon as it is generated, before the client starts buffering video content for a given segment.

Returning to the example introduced earlier, the output of the CTF rate adaptation heuristic in that case comprises six high-quality plus ten low-quality tiles, following the order indicated below in Figure 3. As the diagram illustrates, the client relays this output array to the *Prefetch server*. With this information, the specified tiles are requested concurrently from the *Content server*, taking advantage of a high-capacity link between them. Then, the corresponding video files are loaded into the cache memory, which serves the forthcoming requests from the client with low latency. This in turn should lead to an increase in the bandwidth perceived by the client, and as consequence, also in the quality of the content requested for subsequent video segments.

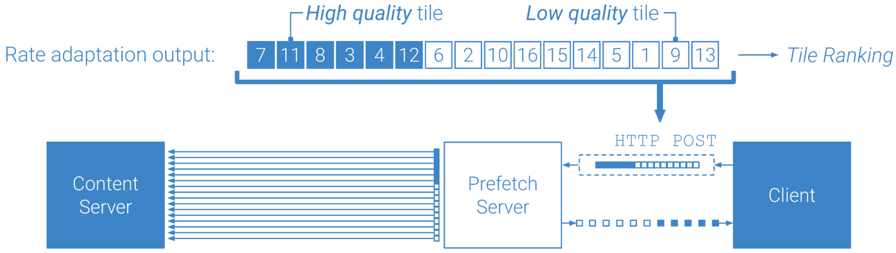


Fig. 3: VR content prefetching: The output of the rate adaptation algorithm is fed to the prefetch server before the client's buffer starts filling up.

Clearly, conducting such a prefetching procedure for each individual user would entail a misuse of the cache memory resources and a substantial increase in the backhaul traffic and the content server's load. To address this issue, we propose a stream processing method for estimating the most salient tiles according to the viewers fixation patterns, on a per-video segment basis. Said set of per-segment salient tiles is then stored into the data structure we refer to as DCoB.

### 3.2 Dynamic Collective Buffer (DCoB)

The DCoB can be understood as a common playout buffer shared by active viewers consuming the same VR content at a certain point in time. Think about the scenario in which a content provider premieres a new episode of a popular show. Many viewers are likely to start a streaming session soon after the episode has been released. In such scenario, clients can benefit greatly from a nearby cache serving content that has been previously requested by other users. Of course, to make the most of the limited memory resources, only a subset of the tiles per segment should be stored into this cache, i.e. those that are most likely to be consumed in ongoing streaming sessions. Arranged in this way, the data in the cache configures a *per-segment collective viewport* or *collective buffer* keeping content from the last $N$ video segments consumed thus far.

This collective buffer has been modeled as a FIFO queue of limited size, backed by a hash table to allow for instantaneous retrieval (see Figure 4 below).

Once the configured capacity is exceeded, the tiles corresponding to the least recently requested segment are evicted, freeing up space in memory for new segments.

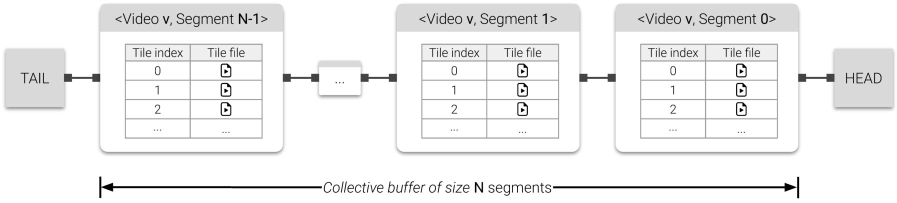


Fig. 4: Collective buffer as a FIFO queue. Each item in the queue corresponds to one segment of a given video and contains the most relevant tiles prefetched from the content server.

The set of video tiles contained within each of the segments of the collective buffer should be dynamically adjusted in response to viewers' fixation patterns. In Section 3.1, a tile ranking was obtained as output of the CTF rate adaptation heuristic. This ordered list of tiles encodes the estimated fixation map of an individual user when watching a particular video segment. In this sense, we have devised an incremental procedure that enables merging the ordered preferences of all the users with an active streaming session, into a single list of video tiles per segment, composing a collective fixation map.

Let us represent a viewer's fixation map for video $v$ and segment $s$ as:

$$\phi_{v,s} = \{\langle t, \rho(t)\rangle : t \in \{1, ..., m \cdot n\}\} \tag{1}$$

where $t$ represents each of the $m \cdot n$ tiles per segment in the tiling scheme $(m \times n)$, while $\rho$ is a function that returns the position in the viewer's tile ranking of the tile passed as argument. Considering the running example from the previous section (see Figure 3), the corresponding fixation map can be expressed in the following terms:

$$\begin{aligned}\phi_{v,s} = \{&\langle 1, 14\rangle, \langle 2, 8\rangle, \langle 3, 4\rangle, \langle 4, 8\rangle, \langle 5, 13\rangle, \langle 6, 7\rangle, \langle 7, 1\rangle, \langle 8, 3\rangle, \langle 9, 15\rangle,\\ &\langle 10, 9\rangle, \langle 11, 2\rangle, \langle 12, 6\rangle, \langle 13, 16\rangle, \langle 14, 12\rangle, \langle 15, 11\rangle, \langle 16, 10\rangle\}\end{aligned} \tag{2}$$

Now, to combine the fixation maps of $K$ viewers watching segment $s$ of video $v$, we start by computing the average position, $\rho$, for each video tile over all $K$-fixation maps. The collective fixation map $(\bar{\phi}_{v,s})$ is defined as follows:

$$\bar{\phi}_{v,s} = \left\{\left\langle t, \frac{1}{K}\sum_{i=1}^{K} \rho_{\langle i\rangle}(t)\right\rangle : t \in \{1, ..., m \cdot n\}\right\} \tag{3}$$

The order of the tiles in $\bar{\phi}_{v,s}$ is determined by their average position, i.e. the smaller this value is for a certain tile, the higher the precedence the tile has for the given video segment.

As stated earlier, only a subset of these tiles should make it to the corresponding segment of the collective buffer. We refer to this subset as *collective viewport*, defined as the *top-k* tiles of the collective fixation map. To determine the value of $k$ we first estimate the correlation between the viewers' fixation maps. High correlation between these maps would imply that users are looking at the same sections of the display, i.e. a few specific tiles. We estimate said correlation by using the *Kendall's tau coefficient* ($\mathcal{K}_\tau$) [41], which measures the correspondence between two ordered sequences in the range $[-1, 1]$: the closer to 1 (resp. $-1$) the higher (resp. lower) the correspondence. Finally, the value of $k$ is set to be proportional to the complement of this correlation coefficient, which we refer to as *Kendall's tau distance* ($\mathcal{K}_{\tau dist}$). Let us take $\bar{\phi}_{{curr}v,s}$ as the current collective fixation map for segment $s$ of video $v$, and $\phi_{{u}v,s}$ as a new fixation map corresponding to user $u$, for the same video segment. The collective viewport size, $k$, is computed as follows:

$$\begin{aligned} k &= \lceil m \cdot n \cdot \mathcal{K}_{\tau dist}(\bar{\phi}_{{curr}v,s}, \phi_{{u}v,s}) \rceil \, ; \\ \mathcal{K}_{\tau dist}(\bar{\phi}_{{curr}v,s}, \phi_{{u}v,s}) &= 1 - \frac{\mathcal{K}_{\tau}(\bar{\phi}_{{curr}v,s}, \phi_{{u}v,s}) + 1}{2} \end{aligned} \tag{4}$$

From the equations in 4, note that in case of perfect correlation ($\mathcal{K}_\tau = 1$), the distance between the fixation maps is zero ($\mathcal{K}_{\tau dist} = 0$), and therefore the viewport size, $k$, is equal to zero as well. In these circumstances, since both the *collective* and *new* fixation maps contain the same collection of tiles, the collective viewport stored into the DCoB for the given segment and video should remain unmodified.

The collective fixation map is incrementally refined as new viewers show up. For this the prefetch server keeps track of the number of viewers ($nViews$) that have watched a given video segment, along with the per-segment cumulative Kendall's tau distance ($agg\mathcal{K}_{\tau dist}$) computed across all the fixation maps received thus far. This data is kept in a key-value store with the tuple $\langle v, s \rangle$ being designated as key:

$$\mathcal{F} : (\langle v, s \rangle) \mapsto \left[\bar{\phi}_{v,s}, nViews, agg\mathcal{K}_{\tau dist}\right] \tag{5}$$

The formal procedure for processing the stream of fixation maps coming from connected VR clients is specified below in Algorithm 1. The process starts by first initializing $\mathcal{F}$ as an empty key-value store (line 7). Then the fixation maps $\phi_{v,s}$ are taken in, one after the other (line 8). Each fixation map updates its corresponding entry on the collective buffer. The *mergeFixationMaps* function in line 16 represents the incremental application of the operation referred earlier in equation 3. The output of this function is the collective fixation map modified by the fixation map being currently processed. The size of the collective viewport, $k$, is determined as the closest integer to the product of the average Kendall's tau distance ($\frac{agg\mathcal{K}_{\tau dist}}{nViews}$) times the total number of tiles ($m \cdot n$). This way the input from previous viewers is weighted and taken into account (line 19). Finally, the tiles belonging to the collective viewport are obtained (i.e. the first $k$ tiles from the collective fixation map), the corresponding video files are retrieved from the *content server*, and the up-to-date data is stored into the collective buffer ($\boldsymbol{DCoB}$) and the key-value store ($\mathcal{F}$) (lines 25-29), after ensuring that the maximum configured capacity ($N$) is not exceeded (lines 21-24).

**Algorithm 1** Dynamic collective buffer

```
1: Let $\mathcal{V}$ be a catalog of 360° videos
2: Let $\boldsymbol{\Phi}$ be a stream of fixation maps forwarded from the VR clients
3: $\boldsymbol{\Phi} = [\phi_{v,s} : v \in \mathcal{V} \wedge s \in \{1, 2, 3, \dots\}]$    ▷ Unbounded set of fixation maps
4: Let $m \cdot n$ be the number of tiles according to the tiling scheme ($m \times n$)
5: Let $N$ the capacity of the collective buffer (number of video segments)
6: Let $\boldsymbol{DCoB}(N)$ be the collective buffer (a FIFO queue backed by a hash table)
7: $\mathcal{F} \leftarrow$ empty dictionary
8: for each fixation map $\phi_{v,s}$ in $\boldsymbol{\Phi}$ do
9:     if $\langle v, s\rangle \notin \mathcal{F}.keys()$ then    ▷ There is no fixation map for tuple $\langle v, s\rangle$ yet
10:        $\bar{\phi}_{v,s} \leftarrow \phi_{v,s}$    ▷ Initialize *collective fixation map*
11:        $nViews \leftarrow 1$
12:        $agg\mathcal{K}_{\tau dist} \leftarrow 0$
13:        $k \leftarrow m \cdot n$    ▷ Initialize size $k$ of the collective viewport to $m \cdot n$
14:    else
15:        $\bar{\phi}_{v,s}, nViews, agg\mathcal{K}_{\tau dist} \leftarrow \mathcal{F}.get(\langle v, s\rangle)$
16:        $\bar{\phi}_{v,s} \leftarrow mergeFixationMaps(\bar{\phi}_{v,s}, \phi_{v,s})$    ▷ Merge $\phi_{v,s}$ into *collective fixation map*
17:        $nViews \leftarrow nViews + 1$
18:        $agg\mathcal{K}_{\tau dist} \leftarrow agg\mathcal{K}_{\tau dist} + \mathcal{K}_{\tau dist}(\bar{\phi}_{v,s}, \phi_{v,s})$
19:        $k \leftarrow \left\lceil \frac{agg\mathcal{K}_{\tau dist}}{nViews}(m \cdot n) \right\rceil$    ▷ Set $k$ to be proportional to average $\mathcal{K}_{\tau dist}$
20:    end if
21:    if $\boldsymbol{DCoB}.size() \geq N$ then    ▷ If the buffer capacity has been exceeded then remove the
   segment at the head of the queue
22:        $\langle v_H, s_H\rangle \leftarrow \boldsymbol{DCoB}.pop()$
23:        $\mathcal{F}.remove(\langle v_H, s_H\rangle)$
24:    end if
25:    $\mathcal{F}.set(\langle v, s\rangle, [\bar{\phi}_{v,s}, nViews, agg\mathcal{K}_{\tau dist}])$    ▷ Update key-value store with new values
26:    if $k \neq 0$ then
27:        $collectiveVP_{v,s} \leftarrow topK(\bar{\phi}_{v,s}, k)$    ▷ Collective viewport set to the top $k$ tiles of $\bar{\phi}_{v,s}$
28:        $VPVideoTiles_{v,s} \leftarrow HTTPGetFromContentServer(collectiveVP_{v,s})$
29:        $\boldsymbol{DCoB}.set(\langle v, s\rangle, VPVideoTiles_{v,s})$    ▷ Update data at the *collective buffer*
30:    end if
31: end for
```

Along with the collective buffer, we also defined a short-lived buffer into which the prefetch server stores the set of *outstanding tiles*, namely those tiles in the viewer's fixation map that remain outside the collective viewport. This in order to avoid the client having to wait for the content server to deliver these tiles during querying time, preventing playout freezes from happening. The entries in this *ephemeral buffer* are volatile and expire over a period of time equivalent to one video segment to minimize their memory footprint. Having both the collective and ephemeral buffers in place ensures that the client can always find relevant content loaded into the prefetch server memory. This way we manage to bypass the *cold-start* problem typical of traditional caching solutions. Figure 5 illustrates a typical sequence of interactions that take place between client, servers and data stores for a single viewer.

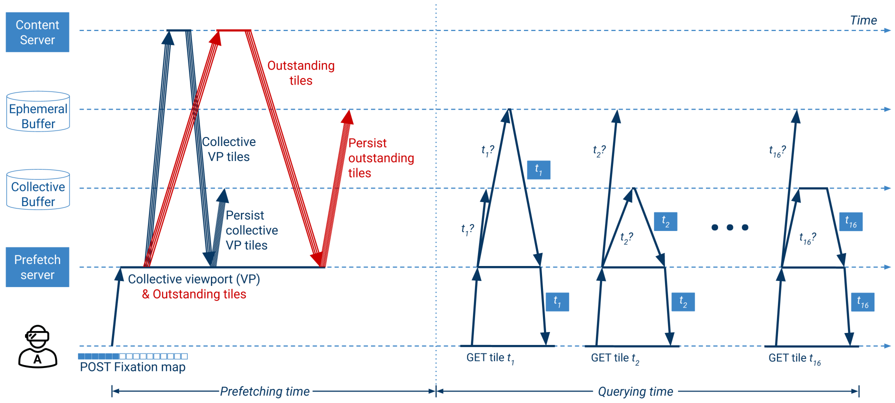


Fig. 5: Timeline of a typical interaction between the entities composing the VR content prefetching approach. At *prefetching time* the collective viewport and outstanding tiles are downloaded into the prefetch server memory. These tiles are served to the client with low latency at *querying time*.

## 3.3 Analysis of computational cost

The procedure in charge of conducting content prefetching has been conceived as an *stateful streaming algorithm* (see Algorithm 1). The input of said procedure consists of regular array structures representing the viewer's fixation maps (consider the example in Equation 2). The length of these arrays is fixed and determined by the number of tiles of the tiling scheme in use, i.e., $m \cdot n$. The proposed algorithm processes each array on an individual basis, and the output of such a processing alters the state of a collective fixation map, and the collective and ephemeral buffers, for a given video $v$ and segment $s$. These data structures represent the state being managed by the algorithm. Let us consider the cost incurred in this procedure both in terms of space and time.

### *3.3.1 Space cost*

As described earlier in section 3.2, the data structures that maintain the state in the proposed algorithm are all arranged into hash tables persisted in memory to allow for fast read and write operations. The hash tables of both the key-value store holding the per-segment collective fixation maps ($\mathcal{F}$), and the collective buffer ($\boldsymbol{DCoB}$) have a fixed capacity in terms of the number of segments they can contain. Said capacity is set upfront via a configuration parameter $N$. In this sense, the space cost due to these two data structures is proportional to $O(N)$.

The hash table backing the ephemeral buffer stores individual tiles which are not part of the collective viewport for a given video and segment. In these circumstances, the space cost is proportional to the number of tiles the viewer is likely to watch in the upcoming segment that fall outside the collective viewport. Said number is never greater than $m \cdot n$ (worst-case scenario). Additionally, the video content persisted in this ephemeral buffer is short-lived by design, which further reduces its memory footprint.

### *3.3.2 Time cost*

At the core of the procedure for maintaining the collective buffer lie two operations:

*i.* the function that updates the collective fixation map ($\bar{\phi}_{v,s}$) for a certain video $v$ and segment $s$, taking in a new unseen fixation map ($\phi_{v,s}$) (see line 16 in Algorithm 1), and

*ii.* the function that incrementally computes the Kendall's tau distance ($\mathcal{K}_{\tau dist}$) between the current $\bar{\phi}_{v,s}$ and the incoming $\phi_{v,s}$ (see line 18 in Algorithm 1).

The first operation consists in computing the element-wise average of two indexed arrays of size $m \cdot n$, and subsequently sorting the resulting array on the obtained values. By using an algorithm such as *mergesort*, the time it takes for this operation to run is proportional to $O(mn \log mn)$.

The Kendall's tau distance in the second operation is computed using the method by Knight W. R. [42], implemented in the `SciPy` Python library. This method is known to have linearithmic time complexity, which in this particular case translates to $O(mn \log mn)$, just as with the above-mentioned operation.

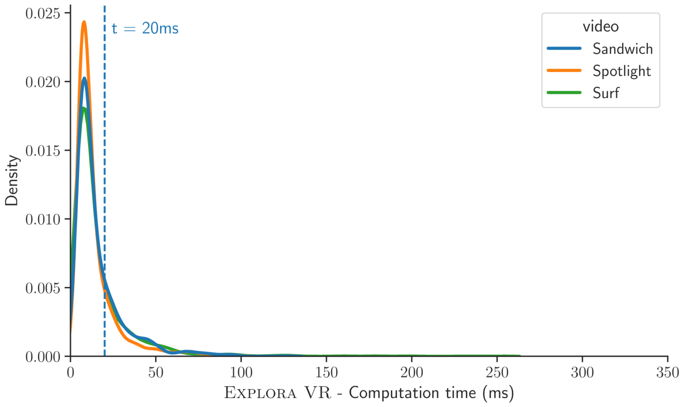


Fig. 6: Experimental estimate of the time required to compute the collective viewport. An in-depth description of the setup is provided in Section 5.1.

Since $m$ and $n$ values are fixed and typically small (consider for instance a $4 \times 4$ tiling scheme), the proposed algorithm is expected to feature a low and fairly consistent execution time. Figure 6 shows an example of the computation times measured on an experimental setting with 48 viewers watching the first 30 segments of three different 360° videos, using a 4 × 4 tiling scheme. In said setting (described in detail later in Section 5.1), the devised operations for computing the collective viewport run under 20 milliseconds 80% of the time. This is only $\frac{1}{50}$ to $\frac{1}{200}$ the video segment length used in tile-based omnidirectional video streaming applications which typically ranges between one to four seconds [16].

Note that the computational cost of the proposed mechanism largely depends on configuration parameters such as the collective buffer capacity ($N$) and the tiling schema ($m \times n$). This suggests that, as the number of users increases, memory use will not surge out of control, and processing time will remain consistent, which accounts for the scalability of our content approach.

## 4 Architecture and Proof-of-Concept Implementation

The system that implements the content prefetching mechanisms we introduced in the previous section adopts an architecture featuring highly configurable containerized components. This system supports the emulation of multiple VR video streaming scenarios—with and without prefetching enabled—under different network and load conditions. A diagram of the components and submodules that make up the system is presented in Figure 7. Next, we address the description of the components of this architecture.

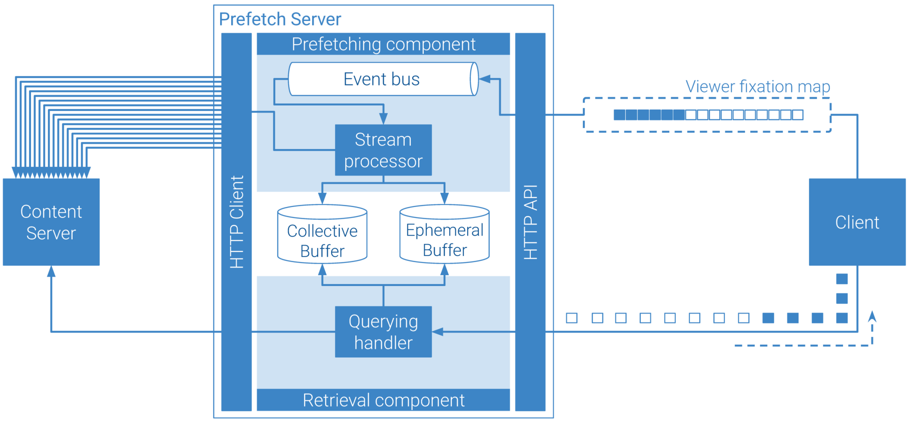


Fig. 7: VR content prefetching architecture: inspired by the Explora framework by Ordonez et al. [43]

### 4.1 Prefetch Server

This is the core component of the system. In devising the functional submodules of this server, we have drawn inspiration from the data processing pipeline presented by Ordonez et al. in [43], which decouples stream data ingestion/preprocessing from data storage and content retrieval. The prefetch server features three main modules: (*1*) the *prefetching component*, (*2*) the *content buffers*, and (*3*) the *retrieval component*.

The *prefetching component* provides an *event bus* which collects the viewers' fixation maps fed by the VR client. A *stream processor* in this component consumes said fixation maps and runs the procedure specified earlier in Algorithm 1 to incrementally build the collective viewports. The stream processor is also in charge of fetching video content from the *Content server*, and does this by issuing multiple concurrent HTTP requests. We relied on the *Publish/Subscribe* pattern readily available in the *Redis* in-memory data store [44] to implement the event bus. As for the stream processor, we implemented it as a Python application running continuously in background, along with the *HTTP API* in charge of handling the interaction with the client.

The video content fetched from the content server by the stream processor is loaded into the *data buffers*. The *collective buffer* hosts the arrangement of video tiles lying inside the incrementally computed viewports, while the *ephemeral buffer* stores the outstanding tiles as defined at the end of Section 3.2. Both buffers are backed by key-value databases implemented in *Redis*.

The *retrieval component* implements the *querying handler* submodule in charge of processing clients' requests for video content. Upon receiving a query, this handler looks up the corresponding video tile file into both the collective and ephemeral buffers. In case the video file is not available yet in none of the prefetch buffers (e.g., due for instance to quality mismatch or network delay), the handler would relay the request to the content server.

The implementation of the prefetch server is available online at `https://github.com/LeandroOrdonez/explora-vr-cache`.

### 4.2 Content Server

This component plays the role of one of the nodes from a content delivery network (CDN). The *content server* consists of a containerized Web server publishing the tiled video content through a HTTP API. Video files are served from the local file system of this component in response to regular `HTTP/1.1 GET` requests matching the following the URL pattern:

```
http://<:host>:<:port>/<:video_id>/<:t_hor>x<:t_vert>/
    <:quality_id>/seg_dash_track<:tile_id>_<:segment_id>.m4s
```

where `t_hor` and `t_vert` stand for the number of tiles in the horizontal and vertical axes respectively, according to the applied tilling scheme.

This content server component was implemented as a Python Web application using the *Flask* framework and *NGINX+uWSGI* as application server. The code of this implementation is available online as well at `https://github.com/LeandroOrdonez/explora-vr-server`.

### 4.3 Client

This component is a containerized adaptation of the headless Virtual Reality client developed by van der Hooft et al. [32]. The headless VR client is an adaptive streaming application written in Python which is able to recreate video streaming sessions from prerecorded head movement traces. By deploying this component as an independent containerized application, we were able to spawn multiple concurrent video streaming sessions, allowing us to assess the response of the proposed VR video content prefetching mechanism under different network and load conditions. The code of the original implementation of the headless VR client is available at `https://github.com/jvdrhoof/VRClient`, while our adaptation can be found at `https://github.com/LeandroOrdonez/explora-vr-dash-client`.

## 5 Experimental Evaluation

To determine the strengths, costs and limitations of the content prefetching mechanism, we have conducted a benchmark evaluation on various VR video streaming setups, with and without the prefetch mechanism in place. The prefetching approach presented in this article was also compared to a caching strategy with a traditional *least-recently used* (LRU) replacement policy which is a common baseline used for evaluating the performance of existing edge-assisted solutions. A description of the environment configuration and the covered test scenarios is presented next, along with the results obtained from this evaluation.

### 5.1 Experimental Setup

The experimental testbed we used in this evaluation is depicted in Figure 8. Each of the components in this diagram were deployed as an isolated *Docker* container, running on a single host machine with 20GB RAM, Intel E5645s @ 2.4GHz processor, and 54GB Hard Disk, using the infrastructure provided by the *imec/IDLab Virtual Wall* environment [45]. As is typically the case, we assume the link between the content server and the cache-enabled edge server to have higher capacity/higher latency than the one between the prefetch server and the VR clients. To emulate these conditions, we have run *traffic control* (`tc`) [46] on each of the containers. This way, we have provisioned a connection between content and prefetch server with 1 Gbps bandwidth capacity and 25 milliseconds latency. On the client's end, we set the latency to 5 milliseconds for the setup with prefetching enabled, and 30 milliseconds in the setup without prefetching—i.e. we kept the same round trip time (RTT) between client and content server in both setups. We gradually increased the bandwidth in the clients link from 10 Mbps to 50 Mbps, and estimated the impact the devised prefetching mechanism has on the quality of experience

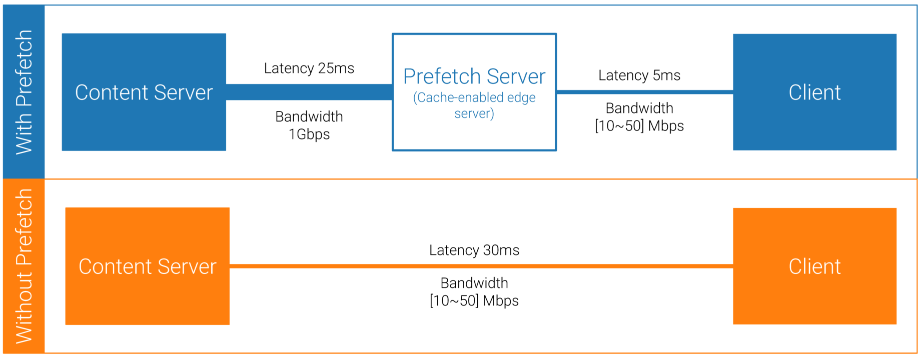


Fig. 8: Experimental testbed for evaluating the VR content prefetching mechanism

(QoE) perceived by the user, measured in terms of delivered video quality, startup delay, and occurrence of playout freezes, as reported by the VR client on a per-segment basis. Finally, these results are contrasted to those obtained from a setup implementing a traditional LRU replacement policy in the cache-enabled edge server.

As for the video content we used the dataset created by Wu et al. [21], which provides head movement traces recorded from 360° video streaming sessions. This dataset comprises the traces collected from 48 unique users while watching nine different VR videos. The tests run in this evaluation consider three representative videos out of the original nine: *Sandwich* features a fragment of a talk show in which most of the motion concentrated in the center of the display; *Spotlight* presents a more dynamic sequence typical for an action movie; *Surf* displays a compilation of video clips recorded with a GoPro camera in an open environment. A tiling scheme of $4 \times 4$ was applied to each of these videos at 4K resolution and 30 FPS, using the same encoder and parameters discussed in [32] and listed in Table 2. We used two quality levels to encode each of the three videos, corresponding to constant rate factors (CRF) of 15 (*High quality*) and 35 (*Low quality*). Table 3 summarizes the resulting bitrates for both quality representations.

Table 2: Overview of encoding parameters

| **Parameter** | **Value** |
|---|---|
| Encoder | HEVC Test Model (HM) |
| Tiling scheme | 4×4 at 4K resolution and 32 FPS |
| GOP | 32 |
| Segment duration | ≈1.067s |
| CRF | [15, 35] |

Table 3: Quality levels and corresponding bitrates for the three videos.

| Video | Bitrate [Mbps] | |
|---|---|---|
| | High Quality | Low Quality |
| Sandwich | 21.9±6.6 | 1.2±0.3 |
| Spotlight | 20.8±13.9 | 1.4±1.3 |
| Surf | 26.4±12.7 | 2.4±1.4 |

With this setup in place, we proceeded to emulate a scenario with multiple users connecting to a video streaming event. In this scenario, each of the 48 viewers in the dataset by Wu et al. would start a streaming session to watch the first 30 segments—this is 32 seconds for a segment duration of 1.067 seconds—of each of the three considered videos. In order to approximate the dynamics of such *near-live on demand* streaming scenario serving multiple users, we set up the experiment so that viewers arrive to their watching session in quick succession with a 5 second separation between each other. This means there were no more than six users watching the same video at a given time.

We have run this simulation for three different configurations: (*i*) `NO_PREFETCH`: no prefetching/cache enabled, (*ii*) `PREFETCH`: prefetching enabled with a collective buffer of 30 segments in size, and (*iii*) `LRU`: caching with LRU replacement strategy and cache size limited to 70MB, which is slightly above the maximum value of memory used by the prefetching mechanism throughout the experiment, as shown below in Table 4. For each configuration, we measured the performance of the system in terms of segment download time, user's QoE (i.e., video quality, startup time, and occurrence of playback freezes), network traffic between content and edge server, and accuracy of the prefetch buffer/-cache.

Table 4: Memory consumed by the prefetching and caching strategies

| **Bandwidth client's link (Mbps)** | **Edge Server memory use (MB)** | |
|---|---|---|
| | `LRU` | `PREFETCH` |
| 10 | 70 | 57.91 |
| 15 | 70 | 59.25 |
| 20 | 70 | 61.11 |
| 25 | 70 | 62.69 |
| 30 | 70 | 67.04 |
| 35 | 70 | 64.01 |
| 40 | 70 | 67.30 |
| 45 | 70 | 67.46 |
| 50 | 70 | 66.29 |

Finally, the versions of the software tools used in this evaluation are listed in Table 5.

Table 5: Versions of the software used in the experimental setup.

| **Software** | **Version** |
|---|---|
| Docker | 20.10.6, build 370c289 |
| Docker Compose | 1.17.1 |
| Operating System | Ubuntu 18.04.4 LTS |
| Redis server | 5.0.3 |
| NGINX (*content and prefetch* servers) | 1.14.2 |
| uWSGI (*content and prefetch* servers) | 2.0.17.1 |
| Flask (*content and prefetch* servers) | 1.0.2 |

### 5.2 Results

The playout buffer size in VR video streaming is limited to a few segments to allow for fast adaptation to viewport changes. In this sense, these kind of streaming applications are particularly susceptible to buffer starvation and playout freezes. In a setup with a cache-enabled edge server placed between clients and the content server, the rate adaptation heuristic might be tricked into believing that content is closer than it actually is, which leads it to request video tiles in high-quality representations. In case of *cache misses* (i.e. the requested content is not found in the cache's memory) the request has to be relayed back to the server, which entails additional processing and network latency. In said cases, the segment download time might take longer than the segment playback duration. When such conditions persist for several segments during a watching session, buffer draining-out and playout freezes are bound to happen.

Figure 9 shows the empirical cumulative distribution function (ECDF) of the per-segment download time for the three configurations under evaluation (NO_PREFETCH, PREFETCH, and LRU), measured for multiple values of bandwidth on the client's end. Note that both the setup with the proposed prefetching mechanism, as well as the one with the LRU cache replacement policy manage to keep download times under the segment duration limit (SEG_DUR line in Figure 9) for most of the segments across all bitrates and videos. However, for the LRU setup, there is in general a larger proportion of segments taking longer to download than the segment duration: on average 14% of the segments in the LRU configuration, compared to only 7.6% of the segments in the PREFETCH setup. As the capacity on the client's link increases, those segments can take as much as 3.9 seconds to download, which is far higher than the comparable download times from the PREFETCH setup which do not surpass 1.8 seconds in any of the cases. This signals a higher likelihood of cache misses for the LRU configuration, and a more frequent occurrence of playout freezes in this setup, specially for large values of bandwidth on the client's connection.

The foregoing is confirmed by measuring the number and duration of the playout freezes by streaming session. Figure 10 reports on these measurements as a function of the client's bandwidth, for each of the considered videos. Note that the setup with the proposed prefetch mechanism offers a *freeze-free* playback experience to the user, in contrast to the LRU counterpart. According

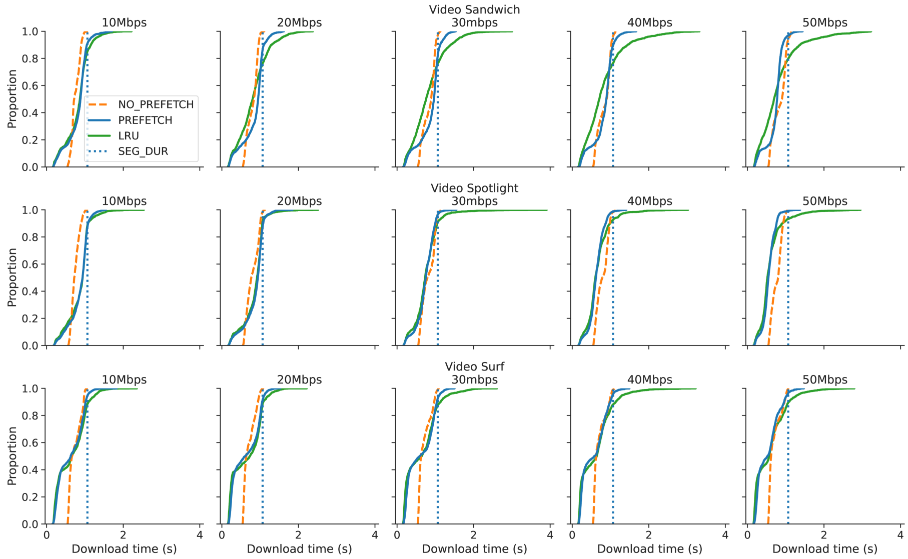
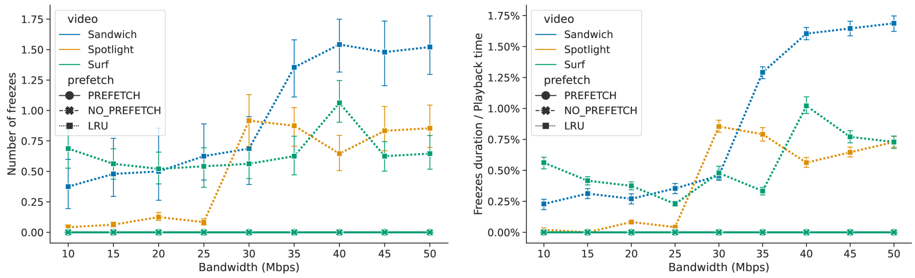


Fig. 9: ECDF of the per-segment download time for the three tested configurations. The larger the number of segments taking longer than SEG_DUR to download, the more likely playout freezes are to occur.



(a) Average number of freezes per watching session.

(b) Average freeze duration to playback time ratio per streaming session.

Fig. 10: Occurrence and duration of playout freezes: The PREFETCH and NO_PREFETCH configurations manage to deliver a freeze-free watching experience to the viewer. For the LRU setup, both frequency and duration of playout freezes increase as the bandwidth on the client's link grows larger.

to Figure 10a, the average number of freezes per streaming session on the LRU configuration is always greater than zero, and the number increases for the three videos as the bandwidth grows larger. We can observe a similar behavior for the total freeze duration. Figure 10b presents this measurement as a proportion of the length of a streaming session, i.e. 32 seconds. These results are clearly inconvenient and counterintuitive from the client's perspective, and can be attributed to the occurrence of cache misses. Table 6 below shows the cache hit ratio measured across the streaming sessions of all 48 users in the dataset, for both LRU and PREFETCH configurations. For the setup with content prefetching enabled, the hit ratio stays above 98% through the entire range of bandwidths, while for the configuration with LRU cache replacement it consistently decreases from 94.6% to 87% as the bandwidth increases. As the bandwidth in the client's link grows larger, the quality of the requested content tends to increase, as does the size of the video tiles stored in the cache. In these circumstances, the LRU cache is only able to accommodate a few items, which in consequence increases the frequency of eviction cycles and cache misses.

Table 6: Hit ratio for different values of bandwidth in the client's link.

| **Bandwidth client's link (Mbps)** | **Hit ratio** | |
|---|---|---|
| | LRU | PREFETCH |
| 10 | 94.62% | 98.44% |
| 15 | 93.84% | 98.43% |
| 20 | 92.56% | 98.36% |
| 25 | 91.65% | 98.51% |
| 30 | 89.40% | 98.52% |
| 35 | 88.17% | 98.60% |
| 40 | 87.37% | 98.69% |
| 45 | 87.27% | 98.97% |
| 50 | 87.02% | 99.13% |

Cache misses also occur as a consequence of the *cold-start* problem that affects passive caching strategies such as LRU. Requests issued against a cold cache are likely to be cache misses and therefore result in retrieval from the origin server. This leads to longer startup delays which degrade the QoE mainly for early viewers. Figure 11 presents the startup delay observed across all the streaming sessions as a function of the client's link capacity. Delay values remain relatively invariable as the bandwidth on the client's connection increases for all the considered configurations. Note that for both PREFETCH and LRU setups, the majority of the values are clustered around 200 milliseconds approximately. That represents a reduction of nearly 3× the startup delay viewers experience in the setup without prefetching/caching enabled. However, a large number of outliers is observed for the LRU configuration lying beyond the segment duration limit. This indicates that many viewers would experience more than one second latency from the moment they initiate the streaming session to the moment the video playback starts. These outliers represent the startup delay perceived by the first users. In Figure 12 only the startup latency measured for the group of early viewers is plotted. On average early users in the LRU configuration would observe around 2× and 6× longer delay times compared to viewers in the NO_PREFETCH and PREFETCH setups, respectively. These results show that the proposed content prefetching mechanism is able to bypass the *cold-start* problem and offer not only shorter startup delay times but also a more consistent experience across all viewers compared to the alternative configurations.

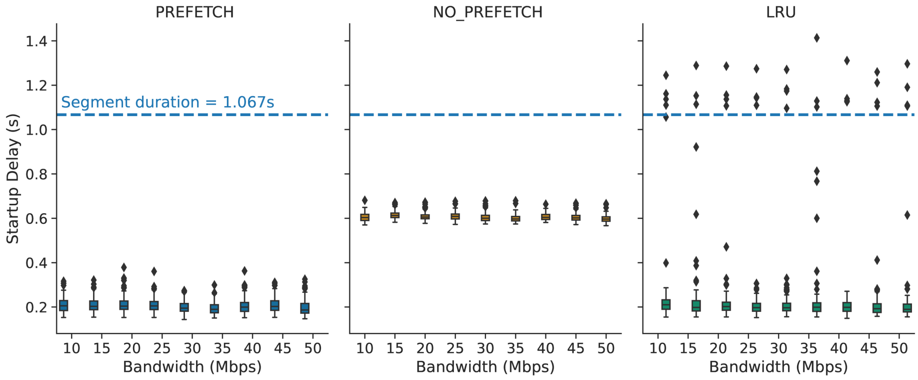


Fig. 11: Startup delay distribution as a function of the client's link capacity. Observations for PREFETCH and LRU configuration are largely concentrated around comparable values. However, outliers for the LRU setup lie farther apart from the bulk of the data, beyond the segment length in many cases. In comparison, the PREFETCH configuration offers a more consistent experience for all viewers.

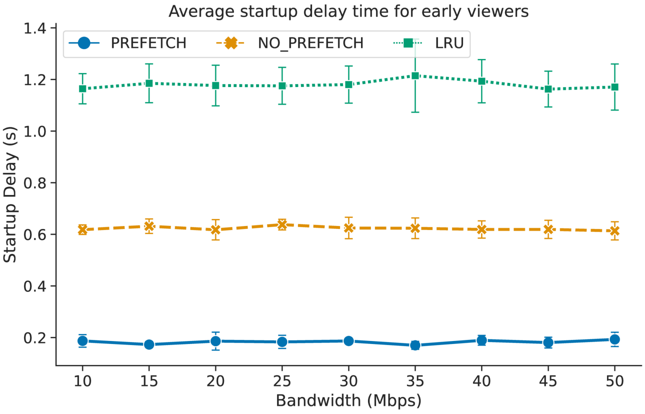


Fig. 12: Startup delay times observed by early users: On average viewers in the PREFETCH configuration would experience 6.5× and 3.4× lower latency than those in the LRU and NO_PREFETCH setups, respectively.

So far, the proposed prefetching mechanism has proven able to deliver a user experience that outperforms that of the alternative setups in terms of segment download time, frequency/duration of playout freezes and startup latency. Let us now look into the perceived video quality. In the HAS client, quality level for each tile in a video segment is determined based on the *perceived bandwidth*, estimated as the quotient between the amount of bits downloaded per segment and the per-segment download time:

$$perceived\_bandwidth(s_i) = \frac{size(s_i)}{download\_time(s_i)} \tag{6}$$

In the expression above, the size of the segment $(s_i)$ is proportional to the quality level of the tiles it comprises. This way, the perceived bandwidth provides a reliable indication of the video quality as observed by the user. Figure 13 shows the average perceived bandwidth over all watching sessions per video, as a function of the actual bandwidth on the client's link. The configuration with content prefetching enabled outperforms the `LRU` setup, most remarkably along the largest values of bandwidth. With the proposed mechanism running on a cache-enabled edge server, clients perceive on average up to 2.5× more link capacity in comparison to the configuration without prefetching, and up to 1.4× compared to the `LRU` configuration. This results in a higher number of tiles being downloaded in high quality.

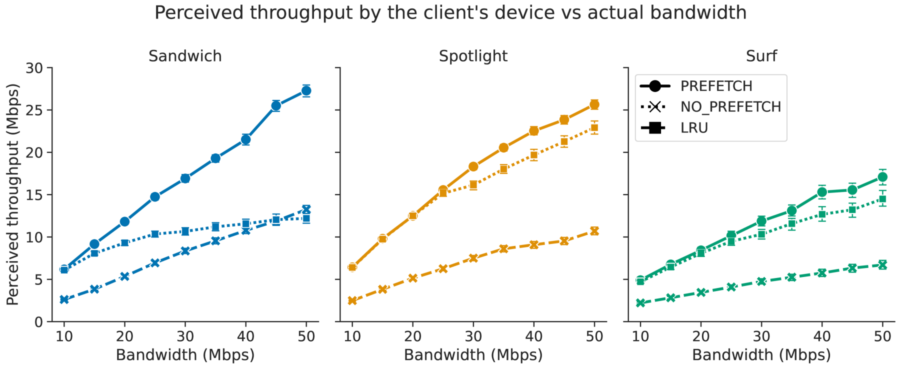


Fig. 13: Client perceived bandwidth as a function of the actual link capacity. User experience greatly benefits from prefetching VR video content into a nearby server.

Figure 14 presents the distribution of the amount of high-quality tiles per segment across the three videos. The mass of the distributions corresponding to each of the setups shifts towards the right (higher number of HQ tiles) as the bandwidth increases. Note that for the configuration with prefetching enabled, the distribution tends to gravitate around 16 tiles/segment at a faster pace than the other two configurations. This proves that across all tested scenarios, the mechanism we propose consistently delivers higher quality of experience for the viewer, compared to the LRU cache alternative, and the plain vanilla client-server configuration.

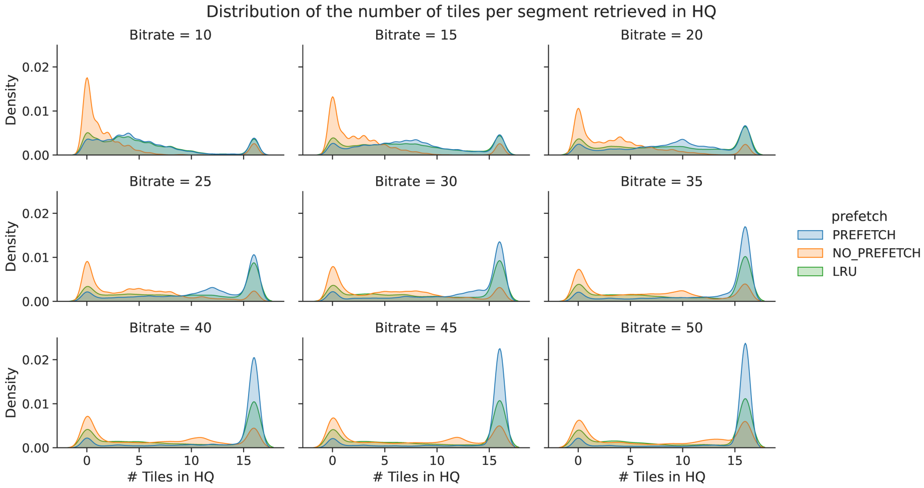


Fig. 14: Distribution of the number of HQ tiles per segment: In comparison to the `LRU` and `NO_PREFETCH` configurations, the number of tiles retrieved in HQ from the prefetch setup increases more rapidly as the bandwidth grows larger. Bitrate values in the charts are in Mbps.

Another appealing effect of prefetching and caching video content into an edge server is the reduction of network traffic to and from the content site. Table 7 presents the network traffic (in gigabytes) measured in the content server interface for the configuration without prefetching/caching enabled, along with the relative change of this metric for the `LRU` and `PREFETCH` setups, and how these measurements vary as the bandwidth on the client's link increases (see Figure 15 for the absolute values). Note that, thanks to the reuse enabled by the LRU and prefetching configurations, there is an important reduction in traffic to the content server: from 75% to 84% for `LRU` caching, and from 36% to 83% for the prefetching server. Also, it is worth noting that the setup with the proposed prefetching mechanism enables these network traffic savings while serving the highest video quality among the tested configurations. That is to say, serving a comparable video quality directly from the content server—without any prefetch/cache capabilitites—would require several times the network traffic reported in Table 7.

To understand why the LRU configuration results in a higher reduction of network traffic with respect to the implementation of the proposed prefetching mechanism, consider the fact that the latter setup is able to consistently deliver higher video quality levels than the former one throughout the entire range of bandwidth values. An increase in the capacity of the client's connection leads to a corresponding increase in the network throughput. This in turn prompts the client to request video tiles in higher quality representations, which consequently drives up the network traffic consumption. Figure 16 portrays the relation between bandwidth at the client side, network traffic in the content server's link, and video quality in terms of the number of high-quality tiles per segment delivered to the client. Note that while the setup with the LRU cache replacement strategy gets the upper hand with regard to network traffic reduction, the enhanced video quality added to the smooth playback offered by the proposed prefetching mechanism, makes for a far superior QoE for the viewer. In this sense, the increase in network traffic to the content server for this configuration can be regarded as a reasonable price to pay.

Table 7: Network traffic between content and prefetch servers for different values of bandwidth in the client's link.

| **Bandwidth client's link (Mbps)** | `NO_PREFETCH` **network traffic (GB)** | **% network traffic reduction** | |
|---|---|---|---|
| | | `LRU` | `PREFETCH` |
| 10 | 1.11 | -75.68% | -36.04% |
| 15 | 1.64 | -78.66% | -44.51% |
| 20 | 2.24 | -80.36% | -48.21% |
| 25 | 2.82 | -82.27% | -53.90% |
| 30 | 3.43 | -80.76% | -64.14% |
| 35 | 3.89 | -81.75% | -64.78% |
| 40 | 4.24 | -82.31% | -67.45% |
| 45 | 4.57 | -83.15% | -80.96% |
| 50 | 5.01 | -84.23% | -83.03% |

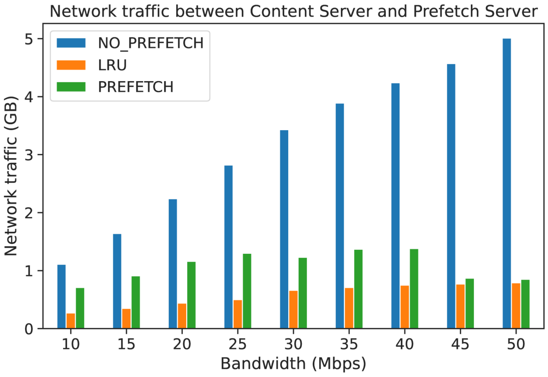


Fig. 15: Network traffic to the *content server* for the three considered configurations, as a function of the bandwidth in the client's link: Both the `LRU` and `PREFETCH` setups manage to induce a notable decline in network traffic. The prefetching mechanism enables this while delivering the highest quality of experience among the considered configurations.

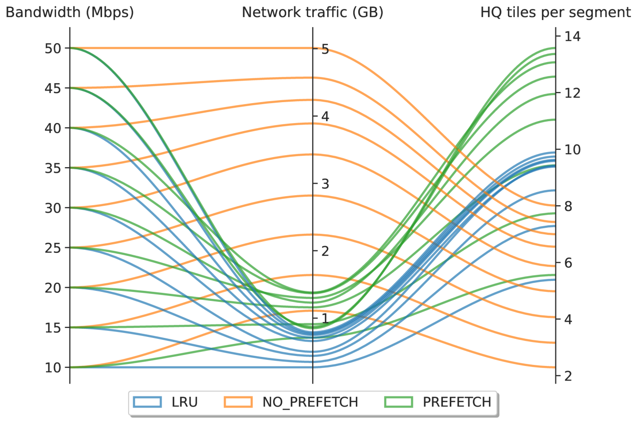


Fig. 16: Relation between client's link bandwidth, network traffic in the backhaul link, and video quality. Both `LRU` and `PREFETCH` setups drive backhaul traffic down while increasing the number of tiles per segment served in high quality. The increase in network traffic use for the `PREFETCH` setup in relation to the `LRU` configuration obeys to a corresponding increase in the delivered video quality.

## 6 Conclusions

Immersive video applications are known for having an immense potential in sectors such as entertainment, education, healthcare, and digital services, among others. However, the existing network infrastructure still struggles to meet the stringent latency and bandwidth requirements of these kind of services, which remains a barrier to enable their broad adoption. In this paper we presented Explora-VR, an edge-assisted solution that allows for low-latency video streaming for tile-based immersive content.

Explora-VR thrives on prefetching the tiles that users are likely to watch in the upcoming segments by advertising the outcome of the viewport prediction and rate adaptation algorithms, before the client starts consuming the content. Prefetched video tiles are downloaded to a cache-enabled edge server located in close proximity to the user, allowing for low-latency content retrieval. This in turn increases the link capacity perceived at the client's end, and in consequence also the quality level of the requested video tiles.

Additionally, the proposed solution supports content prefetching for an *on-demand*, *near-live* scenario, i.e. serving multiple active watching sessions streaming the same content within a narrow time window. To prevent the system from overflowing the content server with duplicate requests while doing this, Explora-VR features a stream processing mechanism that incrementally builds a *collective playout buffer* to serve the requests from active users.

This collective buffer is an in-memory data structure storing a fixed number of *collective viewports*, namely the group of tiles viewers tend to fixate the most on a per-segment basis. The per-segment collective viewports are continuously updated as new viewers arrive to dynamically accommodate to changes in the current preferences from the audience.

We evaluated the performance of Explora-VR against a conventional *client-server* setup with no support for caching or prefetching, and an edge-assisted configuration implementing a regular LRU cache replacement strategy. Our solution proved to be effective in providing a smooth video playback, while also increasing the quality of the delivered content. Under equivalent network conditions, the devised prefetching mechanism leads to an average increase of 2.5× and 1.4× in the effective bandwidth perceived at the client's device compared to the conventional client-server and LRU setups, respectively. This in turn results in a proportional increase in the number of viewport tiles served in high quality. Moreover, in contrast to the alternative LRU configuration, our solution can consistently serve more than 98% of the content requests from the edge server. This means that only a minor proportion of the client requests get relayed to the origin content server, making for a freeze-free playback experience for the user. The foregoing also signals the ability of the proposed approach to bypass the *cold-start* problem that typically affect passive caching strategies. The observed startup delay times show that Explora-VR consistently provides low startup latency for all users including early viewers. These results also hint at the potential of the proposed solution to aid in the recovery from eventual playback freezes. The proximity of the edge server coupled with the high prefetch hit ratio ensures that viewers can quickly resume the playback with a delay we expect to be comparable with the observed startup latency. Additional evaluations with real network traces are needed to confirm this assumption. The devised collective buffer also proved efficient in reducing the load on the content server network. Even though the LRU cache replacement policy outperforms the prefetching mechanism regarding this metric, the superior quality of experience that our approach can offer to the viewer reasonably outweighs this drawback.

In developing Explora-VR, we assumed a number of conditions that will be relaxed in future work to make this solution more suitable for a production-level VR video streaming service. In this sense, further research is going to explore the effect of working with a lossy wireless network in the performance of the content prefetching mechanism. Likewise, the proposed solution will be extended to support multiple intermediate quality representations, instead of only *low-quality* and *high-quality* levels. We expect the results of this work will motivate further studies on edge-assisted prefetching techniques for omnidirectional video streaming.

## References

1. Yépez, J., Guevara, L., Guerrero, G.: AulaVR: Virtual reality, a telepresence technique applied to distance education. In: 2020 15th Iberian Conference on Information Systems

and Technologies (CISTI), pp. 1–5. IEEE (2020)

2. Kwok, A.O., Koh, S.G.: COVID-19 and extended reality (XR). Current Issues in Tourism pp. 1–6 (2020)
3. Singh, R.P., Javaid, M., Kataria, R., Tyagi, M., Haleem, A., Suman, R.: Significant applications of virtual reality for covid-19 pandemic. Diabetes & Metabolic Syndrome: Clinical Research & Reviews **14**(4), 661–664 (2020)
4. Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A.: Edge computing: A survey. Future Generation Computer Systems **97**, 219–235 (2019). DOI https://doi.org/10.1016/j.future.2019.02.050. URL `https://www.sciencedirect.com/science/article/pii/S0167739X18319903`
5. Shi, S., Gupta, V., Hwang, M., Jana, R.: Mobile vr on edge cloud: a latency-driven design. In: Proceedings of the 10th ACM Multimedia Systems Conference, pp. 222–231 (2019)
6. Stauffert, J.P., Niebling, F., Latoschik, M.E.: Latency and cybersickness: Impact, causes and measures. a review. Frontiers in Virtual Reality **1**, 31 (2020)
7. Vega, M.T., Liaskos, C., Abadal, S., Papapetrou, E., Jain, A., Mouhouche, B., Kalem, G., Ergüt, S., Mach, M., Sabol, T., et al.: Immersive interconnected virtual and augmented reality: A 5g and iot perspective. Journal of Network and Systems Management **28**(4), 796–826 (2020)
8. Yaqoob, A., Bi, T., Muntean, G.M.: A survey on adaptive 360° video streaming: Solutions, challenges and opportunities. IEEE Communications Surveys & Tutorials **22**(4), 2801–2838 (2020)
9. He, D., Westphal, C., Garcia-Luna-Aceves, J.: Network support for ar/vr and immersive video application: A survey. In: ICETE (1), pp. 525–535 (2018)
10. Long, K., Cui, Y., Ye, C., Liu, Z.: Optimal wireless streaming of multi-quality 360 vr video by exploiting natural, relative smoothness-enabled and transcoding-enabled multicast opportunities. IEEE Transactions on Multimedia (2020)
11. Zhao, L., Cui, Y., Liu, Z., Zhang, Y., Yang, S.: Adaptive streaming of 360 videos with perfect, imperfect, and unknown fov viewing probabilities in wireless networks. IEEE Transactions on Image Processing **30**, 7744–7759 (2021)
12. van der Hooft, J., Vega, M.T., Petrangeli, S., Wauters, T., De Turck, F.: Optimizing adaptive tile-based virtual reality video streaming. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 381–387. IEEE (2019)
13. van der Hooft, J., Vega, M.T., Petrangeli, S., Wauters, T., De Turck, F.: Tile-based adaptive streaming for virtual reality video. ACM Trans. Multimedia Comput. Commun. Appl. **15**(4) (2019). DOI 10.1145/3362101. URL `https://doi.org/10.1145/3362101`
14. Hosseini, M.: View-aware tile-based adaptations in 360 virtual reality video streaming. In: 2017 IEEE Virtual Reality (VR), pp. 423–424. IEEE (2017)
15. Xie, L., Xu, Z., Ban, Y., Zhang, X., Guo, Z.: 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In: Proceedings of the 25th ACM international conference on Multimedia, pp. 315–323 (2017)
16. Graf, M., Timmerer, C., Mueller, C.: Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In: Proceedings of the 8th ACM on Multimedia Systems Conference, pp. 261–271 (2017)
17. Nguyen, D.V., Tran, H.T., Pham, A.T., Thang, T.C.: An optimal tile-based approach for viewport-adaptive 360-degree video streaming. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **9**(1), 29–42 (2019)
18. Lo, W.C., Fan, C.L., Lee, J., Huang, C.Y., Chen, K.T., Hsu, C.H.: 360 video viewing dataset in head-mounted virtual reality. In: Proceedings of the 8th ACM on Multimedia Systems Conference, pp. 211–216 (2017)
19. David, E.J., Gutiérrez, J., Coutrot, A., Da Silva, M.P., Callet, P.L.: A dataset of head and eye movements for 360 videos. In: Proceedings of the 9th ACM Multimedia Systems Conference, pp. 432–437 (2018)
20. Fremerey, S., Singla, A., Meseberg, K., Raake, A.: Avtrack360: An open dataset and software recording people's head rotations watching 360° videos on an hmd. In: Proceedings of the 9th ACM Multimedia Systems Conference, pp. 403–408 (2018)
21. Wu, C., Tan, Z., Wang, Z., Yang, S.: A dataset for exploring user behaviors in vr spherical video streaming. In: Proceedings of the 8th ACM on Multimedia Systems

Conference, MMSys'17, p. 193–198. Association for Computing Machinery, New York, NY, USA (2017). DOI 10.1145/3083187.3083210. URL https://doi.org/10.1145/3083187.3083210

22. Rossi, S., Ozcinar, C., Smolic, A., Toni, L.: Do users behave similarly in vr? investigation of the user influence on the system design. ACM Trans. Multimedia Comput. Commun. Appl. **16**(2) (2020). DOI 10.1145/3381846. URL https://doi.org/10.1145/3381846
23. Papaioannou, G., Koutsopoulos, I.: Tile-based caching optimization for 360° videos. In: Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Mobihoc '19, p. 171–180. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3323679.3326515. URL https://doi.org/10.1145/3323679.3326515
24. Mahzari, A., Taghavi Nasrabadi, A., Samiei, A., Prakash, R.: Fov-aware edge caching for adaptive 360 video streaming. In: Proceedings of the 26th ACM international conference on Multimedia, pp. 173–181 (2018)
25. Maniotis, P., Thomos, N.: Viewport-aware deep reinforcement learning approach for 360 video caching. IEEE Transactions on Multimedia (2021)
26. Carlsson, N., Eager, D.: Had you looked where i'm looking? cross-user similarities in viewing behavior for 360-degree video and caching implications. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering, ICPE '20, p. 130–137. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3358960.3379129. URL https://doi.org/10.1145/3358960.3379129
27. Dai, J., Zhang, Z., Mao, S., Liu, D.: A view synthesis-based 360° vr caching system over mec-enabled c-ran. IEEE Transactions on Circuits and Systems for Video Technology **30**(10), 3843–3855 (2019)
28. Liu, K., Liu, Y., Liu, J., Argyriou, A., Ding, Y.: Joint epc and ran caching of tiled vr videos for mobile networks. In: International Conference on Multimedia Modeling, pp. 92–105. Springer (2019)
29. Wang, S., Tan, X., Li, S., Xu, X., Yang, J., Zheng, Q.: A qoe-based 360° video adaptive bitrate delivery and caching scheme for c-ran. In: 2020 16th International Conference on Mobility, Sensing and Networking (MSN), pp. 49–56. IEEE (2020)
30. Chang, Z., Lei, L., Zhou, Z., Mao, S., Ristaniemi, T.: Learn to cache: Machine learning for network edge caching in the big data era. IEEE Wireless Communications **25**(3), 28–35 (2018)
31. Petrangeli, S., Swaminathan, V., Hosseini, M., De Turck, F.: An http/2-based adaptive streaming framework for 360° virtual reality videos. In: Proceedings of the 25th ACM International Conference on Multimedia, MM '17, p. 306–314. Association for Computing Machinery, New York, NY, USA (2017). DOI 10.1145/3123266.3123453. URL https://doi.org/10.1145/3123266.3123453
32. van der Hooft, J., Torres Vega, M., Petrangeli, S., Wauters, T., De Turck, F.: Quality assessment for adaptive virtual reality video streaming: A probabilistic approach on the user's gaze. In: 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), pp. 19–24 (2019). DOI 10.1109/ICIN.2019.8685904
33. Qian, F., Ji, L., Han, B., Gopalakrishnan, V.: Optimizing 360 video delivery over cellular networks. In: Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges, ATC '16, p. 1–6. Association for Computing Machinery, New York, NY, USA (2016). DOI 10.1145/2980055.2980056. URL https://doi.org/10.1145/2980055.2980056
34. Xu, Z., Zhang, X., Zhang, K., Guo, Z.: Probabilistic viewport adaptive streaming for 360-degree videos. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE (2018)
35. Zhang, Y., Zhao, P., Bian, K., Liu, Y., Song, L., Li, X.: Drl360: 360-degree video streaming with deep reinforcement learning. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1252–1260. IEEE (2019)
36. Vielhaben, J., Camalan, H., Samek, W., Wenzel, M.: Viewport forecasting in 360° virtual reality videos with machine learning. In: 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), pp. 74–747. IEEE (2019)
37. Fan, C.L., Lee, J., Lo, W.C., Huang, C.Y., Chen, K.T., Hsu, C.H.: Fixation prediction for 360 video streaming in head-mounted virtual reality. In: Proceedings of the 27th

Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 67–72 (2017)

38. Zhu, Y., Zhai, G., Min, X.: The prediction of head and eye movement for 360 degree images. Signal Processing: Image Communication **69**, 15–25 (2018)
39. Sitzmann, V., Serrano, A., Pavel, A., Agrawala, M., Gutierrez, D., Masia, B., Wetzstein, G.: Saliency in vr: How do people explore virtual environments? IEEE transactions on visualization and computer graphics **24**(4), 1633–1642 (2018)
40. Chen, X., Kasgari, A.T.Z., Saad, W.: Deep learning for content-based personalized viewport prediction of 360-degree vr videos. IEEE Networking Letters **2**(2), 81–84 (2020)
41. Prematunga, R.K.: Correlational analysis. Australian Critical Care **25**(3), 195–199 (2012)
42. Knight, W.R.: A computer method for calculating kendall's tau with ungrouped data. Journal of the American Statistical Association **61**(314), 436–439 (1966)
43. Ordonez-Ante, L., Van Seghbroeck, G., Wauters, T., Volckaert, B., De Turck, F.: Explora: Interactive querying of multidimensional data in the context of smart cities. Sensors **20**(9), 2737 (2020)
44. Gutierrez, F.: Messaging with redis. In: Spring Boot Messaging, pp. 81–92. Springer (2017)
45. imec/IDLab: Virtual wall: Perform large networking and cloud experiments. (2021). URL `https://doc.ilabt.imec.be/ilabt/virtualwall/index.html`
46. Brown, M.A.: Traffic control howto. Guide to IP Layer Network **49**, 36 (2006)