Temporal Characterization and Prediction of VR Traffic: A Network Slicing Use Case

Federico Chiariotti, *Member, IEEE*, Matteo Drago, *Member, IEEE*, Paolo Testolina, *Student Member, IEEE*, Mattia Lecci, *Member, IEEE*, Andrea Zanella, *Senior Member, IEEE*, and Michele Zorzi, *Fellow, IEEE*

***Abstract*—Over the past few years, the concept of Virtual Reality (VR) has attracted increasing interest thanks to its extensive industrial and commercial applications. Currently, the 3D models of the virtual scenes are generally stored in the VR visor itself, which operates as a standalone device. However, applications that entail multi-party interactions will likely require the scene to be processed by an external server and then streamed to the visors. However, the stringent Quality of Service (QoS) constraints imposed by the VR's interactive nature require Network Slicing (NS) solutions, for which profiling the traffic generated by the VR application is crucial. To this end, we collected more than 4 hours of traces in a real setup and analyzed their temporal correlation, focusing on the Constant Bit Rate (CBR) encoding mode, which should generate more predictable traffic streams. From the collected data, we then distilled two prediction models for future frame size, which can be instrumental in the design of dynamic resource allocation algorithms. Our results show that even the state-of-the-art H.264 CBR mode may have significant frame size fluctuations, impacting NS optimization. We then exploited the models to dynamically determine requirements in an NS scenario, providing the required QoS while minimizing resource usage.**

***Index Terms*—Virtual reality, extended reality, traffic modeling, network slicing, resource provisioning.**

I. Introduction

OVER the past few years, the rapid technological development of Head Mounted Devices (HMDs) and the strong push towards the virtual world caused by the COVID-19 pandemic led to an explosion of the eXtended Reality (XR) market, which includes technologies such as VR, Augmented Reality (AR), and Mixed Reality (MR). Recent studies estimate hundreds of millions of users of these technologies in a time span of just 3 years [1], requiring millions of new devices to be developed, produced, and shipped around the world for a business in the order of billions of dollars [2].

While the latest news on the *metaverse* seem to indicate that the fastest growth will be in the entertainment and social media industries, XR is expected to make an impact in a wide variety of scenarios [3], [4]. Interactive design, marketing, healthcare, and employee training are just a few of the proposed use cases, but industrial remote control in manufacturing and agriculture might have the largest impact, allowing human operators to remotely control machines in risky, hard to reach or unsafe environments, through a fully interactive virtual framework.

A common characteristic of all these new applications is their interactive nature: users do not passively receive the information or stream a video, but need to manipulate the environment while maintaining an illusion of presence that requires the application to operate under very strict end-to-end delay constraints [5], [6]. In particular, safety-critical and industrial applications will have even stricter constraints, as the consequences of network impairments can be significantly more serious. *Cybersickness* is another important issue, as a delay over 20 ms between movements and visual and auditory feedback can cause disorientation and dizziness [1], [7].

In order to fulfill these stringent latency requirements over a wireless connection, the application and the network need to cooperate. The *NS* paradigm [8] divides the resources of a 5th Generation (5G) network [9] among different slices, i.e., services or groups of services who share similar QoS constraints, expressed in terms of bitrate, latency, jitter, and traffic statistics [10]. By dividing users and services into distinct logical connections, NS limits the stochastic effect of cross-traffic, guaranteeing each service a certain amount of resources, according to its Service Level Agreement (SLA) [11], to achieve the desired Quality of Experience (QoE). Most works in this area, however, focus on relatively predictable applications. In this setting, the need for predictability in XR traffic becomes extremely important, leading to a resurgence of *quasi*-CBR encoders, which are not used in non-interactive streaming due to the slightly lower overall perceptual quality [12] and picture quality stability [13]. On the other hand, traditional Variable Bit Rate (VBR) encoders may have frame size differences between independent and predictive frames of up to two orders of magnitude, making the problem of allocating network resources extremely hard. In particular,

Manuscript received 29 December 2022; revised 7 April 2023; accepted 29 May 2023. Date of publication 5 June 2023; date of current version 4 April 2024. The work of Federico Chiariotti was supported in part by European Union under the Italian National Recovery and Resilience Plan of NextGenerationEU, as part of the "SoE Young Researchers" Grant REDIAL. The work of Mattia Lecci and Paolo Testolina was supported in part by Fondazione CaRiPaRo under Grant "Dottorati di Ricerca" 2018 and 2019, respectively. This work was supported in part by the National Institute of Standards and Technology (NIST) under Grant 60NANB21D127. Recommended for acceptance by Y. Xiao. *(Corresponding author: Federico Chiariotti.)*

Federico Chiariotti is with the Department of Information Engineering, University of Padova, 35131 Padua, Italy, and also with the Department of Electronic Systems, Aalborg University, 9220 Aalborg, Denmark (e-mail: federico.chiariotti@unipd.it).

Matteo Drago, Paolo Testolina, Mattia Lecci, Andrea Zanella, and Michele Zorzi are with the Department of Information Engineering, University of Padova, 35131 Padua, Italy (e-mail: dragomat@dei.unipd.it; testolina@dei.unipd.it; leccimat@dei.unipd.it; zanella@dei.unipd.it; zorzi@dei.unipd.it).

This article has supplementary downloadable material available at https://doi.org/10.1109/TMC.2023.3282689, provided by the authors.

Digital Object Identifier 10.1109/TMC.2023.3282689

resource allocation algorithms for this type of encoder would require an explicit exchange of information between the application and the network to perform well, requiring additional signaling protocols to enable cross-layer functionalities.

While some efforts have been devoted by prominent standard bodies on this topic [5], [6], the current availability of traffic models for XR is scarce. Furthermore, to the best of our knowledge, no detailed analysis of the temporal statistics of *quasi*-CBR video streams can be found in the literature, making existing slicing schemes rely on uncertain foundations. This makes the definition of a SLA for XR traffic more complex: most NS solutions assume that each application's demand in terms of required throughput and latency is known, but such a characterization may be difficult in case of streams with variable frame size, requiring significant overprovisioning.

However, *quasi*-CBR encoders are not perfect, and the interplay between the video content and the movements and actions of the users may cause significant fluctuations. In this work, we analyze the traffic from a real VR application using the *Periodic-Intra Refresh* mode of the H.264 codec, which results in relatively small differences in the frame sizes. Recent analyses of Oculus Quest VR traffic have shown that similar settings are used in what is perhaps the most common commercial VR platform [14]. Modeling these imperfections and, consequently, predicting the size of future frames in advance can be extremely important in the allocation of network resources, particularly if some critical QoS constraints, such as maximum latency and jitter, have to be met [15]. Furthermore, the recent *Cloud XR* trend pursued by some major players in the telecommunications industry [16], [17] pushed the latency and throughput requirements even further, as the processing and rendering steps of the XR content are moved from the user's local network to the Cloud.

This increases the need for a dynamic SLA that can allow an NS system to provide low-latency service to XR applications without wasting too many resources: most slicing schemes in the literature define static SLAs that specify a constant required bitrate, which would be fine for perfectly CBR video (in which all frames have the same size, and Motion-To-Photon (MTP) latency becomes deterministic with slicing), but may result in poor performance if frames have different sizes. While *quasi*-CBR encoders do not have the extreme, order-of-magnitude differences in frame sizes common to traditional VBR encoders, the only way to strictly bound the MTP latency is to overprovision resources to accommodate the largest frames. While static overprovisioning is wasteful, as it requires allocating additional resources to all frames, having a temporal model of the XR traffic stream could allow the Base Station (BS) to predict the future needs of the application, tailoring the QoS requirements in the SLA to what will actually be needed.

Hence, in this paper we address the problem of providing a stochastic characterization of a real VR traffic source, so as to allow for a dynamic provisioning of bandwidth resources for VR users to satisfy the latency constraints. Our analysis can also be applied to the downlink part of generic XR traffic, and the basic statistical methods could theoretically be applied to any type of interactive content, although the considerations and the performance of the proposed schemes would depend on the specific features of the source application.

Building upon our previous works [18], [19], in which we collected more than 4 hours of live sessions and performed basic traffic characterization, in this paper we take the analysis one step further by modeling the size of VR frames in the stream as a correlated time series, that is then used to derive an adaptive and predictive SLA. The contributions of this paper are the following:

- We propose two parametric regression models to predict the size of future frames, and show that these models can be generalized to other traces and even different applications with limited regression performance loss;
- We analyze the residual error of these predictors, providing a full statistical model of future frame sizes;
- We show that the prediction can be successfully used for efficient resource allocation in an NS scenario in which a BS must allocate Radio Access Network (RAN) resources to provide high-quality service to a Cloud VR flow streamed from a remote server to the user's HMD;
- We consider different NS modes, including per-user or application-level slicing, and compare the performance of different schemes in terms of the trade-off between resource utilization and latency.

A partial version of this work was presented in [20], where we empirically proved the *quasi*-CBR nature of VR traffic flows. This work significantly extends it by exploring the statistical analysis of frame sizes at a deeper level, including the characterization of the residual error of the predictors, and expanding on the SLA definition, including the use case with multiple VR users. All our traces, as well as the analysis and simulation code, are publicly available.[1]

The rest of the paper is structured as follows. Section II will discuss the current state of the art on the modeling of XR traffic sources and NS, and our experimental setup is briefly presented in Section III. Our analysis is reported in Section IV, while Section V illustrates how our analysis can be leveraged for a simple NS use case by designing predictive resource allocation mechanisms and testing their performance in a simple simulation scenario. Finally, Section VII draws conclusions and presents some avenues for future work.

II. State of the Art

Despite a steady scientific interest in VR since the 1990s [21], relatively little work has been done to characterize the details of this type of traffic, and its possible consequences on resource allocation schemes. In this section we provide an overview of what has been done in the literature to try to fill these gaps, while also providing the key concepts needed to understand why NS could be a powerful to enable XR scenarios.

[1] The VR traffic traces and code are available in this repository: https://github.com/signetlabdei/vr-trace-analysis

A. Motion-to-Photon Latency and VR Sickness

The MTP latency is defined as the time difference between the beginning of a movement of the user's head and the instant when the image that corresponds to the user motion is shown on the HMD screen. This phenomenon is one of the main factors causing sickness when experiencing XR content, the main symptoms being discomfort, nausea, cold sweating, eye fatigue, and disorientation. From a research point of view, a lot of effort has been devoted to avoiding such episodes in the first place, and the IEEE issued a dedicated standard in 2021 [22], which addressed the content design, sickness assessment and measurement, and the network requirements that may influence the MTP latency.

First of all, measuring the MTP latency represents a challenge *per se*. The architecture described in [23], which consists of a control PC, a head position model-based rotary platform, a pixel luminance change detector which converts the change in the display into a voltage value, and a digital oscilloscope to show it, is used as reference in [22]. Specifically, after a movement of the rotary platform is generated by the control PC, the MTP latency is measured as the time difference between the platform's movement and the corresponding voltage change on the oscilloscope.

To obtain a robust estimate of the MTP latency, a precise head tracking algorithm is of the utmost importance. The authors of [24] presented a 6 Degrees of Freedom (6DoF), optical head tracking instrument with a declared motion-to-pose latency (i.e., the time between a change in the user's pose and the tracker actually detecting the movement) of about 28 μs, at a sample rate of 50 kHz. They also showed that the difference between the tracker's pose output and the user's true pose is dependent on pose velocity, tracker sampling rate, tracking latency, and noise. Moreover, the authors of [25] showed that latency jitter artefacts already occur with a low system load by injecting artificial latency in a VR simulation. Even though their hypothesis included the tracking algorithm of the HMD as a possible cause of such jitter spikes, they did not prove it empirically.

Both rotational and translational MTP latencies were estimated in [26] by calculating the phase shift between the captured signals of the physical motion of the HMD and a motion-dependent gradient stimulus rendered on the display. They were able to conclude that rapid head movements may elicit stronger disorientation to users in VR environments than slower head movements do. Even though the measurements were carried out with an Oculus Rift DK2, the proposed methodology is general and can be applied to other HMDs as well. The authors of [27] also measured the MTP latency with different workloads (determined by the complexity of the scene to render), finding that it can span from a minimum of 45 ms to a maximum of 155 ms. In general, the network requirements defined by the standard [22] are way more stringent: approximately 5 ms for the wireless transmission and 20 ms in total for the MTP latency, with a jitter strictly lower than 5 ms.

B. XR Traffic Characterization

XR traffic modeling is closely related to 2D video content, and, even more so, to live, interactive applications such as video conferencing and gaming. However, most of the work on the subject has considered the customary encoding schemes for pre-recorded video streaming, i.e., the VBR encoding based on either the H.264 or the H.265 standard [28]. VBR can provide a stable visual quality, improving the user's QoE [29], but is also subject to significant jitter due to the large frame size fluctuations. Transmitting VBR videos with low latency can then be a significant challenge even over channels with constant capacity [30]. On the other hand, CBR encoding sacrifices some visual quality stability to obtain an encoded video stream with a stable transmission rate [31]. Although the higher predictability of the encoded output makes CBR encoding attractive for interactive video and XR content, it is still relatively unexplored in the relevant literature. However, a recent black box study on Oculus traffic [14] shows that frame size distributions from Cloud gaming traces closely resemble those we obtained with H.264 CBR encoding, suggesting that similar settings are used by state of the art commercial applications as well.

Perhaps the most similar application that has been studied in the literature is *Cloud gaming:* just like in XR streaming, interactive video content is rendered on a remote server and streamed directly to the users without the need for client-side computation. While these applications stream traditional video content instead of binocular, 3D content, some of the requirements in terms of latency and reliability are similar, and the need to address them with optimized protocols and new transmission strategies has led to a significant interest in Cloud gaming traffic characterization, from which we can draw some insights. The authors of [32] carried out an extensive measurement campaign in Google Stadia, a popular Cloud gaming platform, giving an overview of its inner working. They studied the distributions of downlink traffic, packet size and inter-packet time under multiple settings, including different resolutions, video codecs, and network conditions. On the other hand, in [33], [34] direct comparisons were made between different Cloud gaming platforms, mostly focusing only on the bitrate of the video stream, without including latencies or user QoE.

A more comprehensive Cloud gaming testbed, including automated trace acquisition over Ethernet, WiFi, and LTE, was presented in [35]. Automating the acquisitions surely gives an advantage in terms of reproducibility and speed of the experiments, but the unpredictability of the users' actions in gaming scenarios (and, more importantly, in XR) is the real challenge that the network has to face, limiting the usefulness of the results. These works represent a good starting point for the collection and modeling of VR traffic, as it is reasonable to assume that most of these Cloud gaming companies will start providing VR services soon. However, most works still focused on simple applications, such as interactive data visualization [36], and do not provide much insight on more complex scenarios. The extensive literature on immersive video streaming [37] has been mostly focused on passive applications in which the user is only a viewer, with different QoE and encoding considerations. A recent 3GPP report [38] also provided a simple model for XR traffic, which does not consider temporal or video content aspects, and is thus usable for general feasibility studies, but not for fine-grained optimization.

C. Network Slicing

Network Slicing (NS) has been identified as a key technology in 5G and Beyond networks [9], and has been successfully employed in a number of different applications, from the Internet of Things (IoT) [10] to vehicular networks [39].

Network slices are logical entities that allow service separation over the same physical network, tailoring the reserved network resources to the requirements of each service. The NS allocation process can be broadly summarized as follows [10]: first, a network slice is requested, and the required Virtual Network Functions (VNFs) and the connection between them are identified. The resources for the slice are then reserved in the physical network infrastructure, scaling the allocation dynamically according to the variations of the service requirements. The employed VNFs and/or their connection are also flexible, and may be changed seamlessly. Finally, when the slice is no longer required, it is destroyed and its resources are released.

In general, the NS research has focused on the automation of the above process, on the definition of the slices, and on the orchestration and management of the network resources, designing solutions to ensure and enhance the efficiency and the safety of this paradigm. Both these aspects are extremely challenging due the complexity of the underlying network infrastructure and the heterogeneous and dynamic nature of the service requirements [11], [40].

The resource allocation problem has been modeled through classical approaches, like in [41], where a 2-level matching game is formulated with the infrastructure provider as the vendor and the virtual network operator as the buyer, that in turn acts as the vendor to the users. The authors of [42] model the function placement as an integer optimization problem and propose an effective simplification strategy, while [43] defines a path-search allocation strategy to deal with an elastic traffic demand while ensuring reliable communication over a slice. Due to the complexity of the problem, heuristic solutions have also been explored [42].

Recent works have focused on data-driven approaches [44], [45], leveraging the recent advancements of Machine, Deep and Reinforcement Learning techniques. Specifically, assigning the allocation and the orchestration tasks to a controller makes the problem particularly suitable for Reinforcement Learning (RL) approaches. In [46], a centralized controller admits and assigns slices to the users according to their SLAs and traffic usage. The controller takes into account the heterogeneous mobility and traffic models among diverse slices to make decisions. A deep RL approach is proposed in [47] to maximize the welfare of each service provider by jointly allocating communication and computing resources.

D. Prediction-Based Slicing

The use of traffic prediction in NS is a concept that was first explored in [48]: as slicing requires precise SLAs to provide QoS to different services, but most practical applications are VBR, characterizing the traffic and predicting future requests is a way to allocate resources in a foresighted manner, performing resource allocation on a short timescale and admission control on a longer one. If we consider wider networks with massive numbers of users, the daily, weekly, and seasonal cycles of network usage can also be exploited to allocate resources more effectively [49]. This work, however, will focus on shorter-term predictions over a limited number of XR flows. Other works such as [50] have also explored the possibility of exploiting longer-term trends for a rough resource allocation, while using a short-term scheduler for fine-grained optimization.

In particular, the use of Auto-Regressive Moving Average (ARMA) models has been explored in [51] as a potential application-agnostic prediction method to perform resource allocation: as the orchestrator knows the state of the packet buffer for each slice, it can perform the moving average and allocate resources accordingly. However, ARMA models require a certain number of past samples, and this approach cannot discriminate between different applications: consequently, the initial performance will be lower when compared to an application-aware model that takes knowledge of the traffic source into account. In order to capture the behavior of more complex traffic sources, it is also possible to replace the ARMA model with a Long Short-Term Memory (LSTM) deep neural network [52], which can generalize to non-linear and longer-term patterns. Deep reinforcement learning [53] is another alternative, as it can implicitly learn even complex application behaviors and take them into account when slicing.

Another recent idea is to combine NS with video bitrate adaptation: if we consider QoE as a flexible metric over which we can compromise in high traffic conditions, a cross-layer approach allows the orchestrator to dictate the video bitrate for the next few frames [54], limiting the demands of the interactive video flow to what the network is able to support. This approach is complementary to the prediction-based one, as these bitrate changes need to be relatively infrequent to avoid annoying the user, and such a system would operate over a longer timescale: while the prediction and allocation of resources is usually performed over tens or hundreds of milliseconds, video bitrate adaptation spans multiple seconds, and the two approaches can be integrated.

E. XR Resource Management

Although the management of XR flows is a relatively new problem, a few works have already discussed efficient schemes for providing QoS to these applications. For example, in [55], [56] game-theoretic approaches are proposed to tackle the optimization of multi-user VR streaming over a small cell, with the help of machine learning. The authors of [57] analyze the scheduling problem from the perspective of Mobile Edge Cloud (MEC), proposing scheduling strategies and analyzing communication, computing, and caching trade-offs. While the models proposed for the network architectures considered in these works are extremely complex, there is no comparison with real-world VR streaming.

To the best of our knowledge, our previous works, which proposed a simple architecture for collecting traffic traces from VR games [18] and a simple generative model for the frame size [19], were the first to use real VR traffic traces, along with

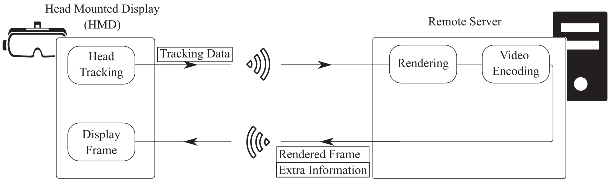


Fig. 1. Basic experimental setup schematic.

the aforementioned passive study of Oculus traces [14]. This paper extends our previous works by characterizing the temporal behavior of the VR traces and drawing novel conclusions for NS optimization.

III. Experimental Architecture

In this section, we describe the architecture of our VR streaming acquisition and give some perspective on the full end-to-end setup. To further understand what are the steps that most influence the VR performance, it is useful to describe a common end-to-end VR architecture. First, we can start from the collection and processing of tracking information, delegated to the HMD. Then, this information is sent to a remote server to compose the viewport, i.e., what is actually shown to the user. This process includes the rendering of the scene, the video encoding providing a more robust transmission towards the mobile device, and possibly some additional information, e.g., the direction in which the rendered frame is supposed to be displayed. After receiving and decoding the video stream together with all the additional meta-information, the HMD generates the images to display at the occurring screen refresh rate.VR The MTP latency, which considers all of these steps, is crucial for determining the user QoE [29], as higher latencies will be perceived by the user and may even cause cybersickness. Recent IEEE [22] and 3GPP [38] standards specify strict MTP latency constraints, and one of the objectives of our modeling effort is to provide a framework for delivering VR content efficiently while respecting these latency requirements.

Our experimental setup consisted of a desktop computer equipped with an NVIDIA GeForce RTX 2080 Ti graphics card acting as the rendering server, and an iPhone XS enclosed in a VR cardboard acting as the HMD. VR applications were thus run on the rendering server and streamed to the headset using the *RiftCat 2.0* application (on the server), and *VRidge* 2.7.7 (on the phone).[2] A basic schematic of the setup is shown in Fig. 1, which includes the main exchanges of information between the desktop computer, acting as a remote server by rendering and encoding the video, and the iPhone XS, which functioned as the HMD. The setup was purposefully simple, with a dedicated WiFi network with enough capacity to maintain a constant visual quality and model the VR source without any effect of rate adaptation or long delays. The behavior of VR applications over connections with time-varying capacity would be determined by a number of factors, e.g., the employed rate adaptation scheme, that have been extensively studied in the literature and are outside the scope of this work, where we focus on deriving a channel-agnostic application model. In the following, we will describe the encoding and communication features of the application in more detail.

The application uses hardware-accelerated H.264 encoding via Nvidia Encoder (NVENC) as long as a compatible graphics card is available to the system. At the time of our experiments, the newer H.265 standard was not supported by the application, as even commercial visors have only recently started supporting the newer encoding standard due to its computational complexity. However, most of the considerations in the following would be the same for H.265 content, although with a higher picture quality at the same bitrate. *RiftCat*'s developers disclosed that Periodic Intra-Refresh is used, a setting provided by the encoder that allows each frame to be roughly the same size, making the stream almost CBR and thus easier to handle from a network perspective. It does so by replacing key frames with *waves* of refreshed intra-coded blocks, i.e., blocks without any dependence on other frames, effectively spreading a key frame over multiple frames. Image quality is balanced with resilience to packet loss by setting the `intraRefreshPeriod`, a parameter that determines the period after which an intra refresh happens again, and the `intraRefreshCnt` parameter, which sets the number of frames over which the intra refresh happens [58]. If we consider a 30 Frames per Second (FPS) video, a value of 30 for the `intraRefreshPeriod` would ensure that the frame is fully recovered every second. On the other hand, choosing small values of `intraRefreshCnt` leads to a quicker refresh but lower quality.

Detailed information about the video encoder is fundamental for our work, since different encoders typically behave differently, especially when analyzing the temporal behavior of the encoded source. Still, we believe that our work offers network researchers a peek into the intricacies of this topic, showing some key results on how a VR traffic flow can be analyzed for resource provisioning.

Different freely available games and applications were used to acquire our dataset, including *Minecraft*, *Virus Popper*, and *Google Earth VR*. Further details on the acquisition setup and our traces can be found in [19]. In the following, we will mostly concentrate on one trace acquired using the *Virus Popper* application, but the methodology applies throughout the dataset, and can be easily replicated for any of the other traces.

IV. Video Trace Analysis

By analyzing the acquired traces, we determined that the application used User Datagram Protocol (UDP) over IPv4. It also used an additional application-layer protocol header of variable size, which we decoded to determine the types of the exchanged packets. More specifically, synchronization and acknowledgment packets were exchanged in both directions, while the Uplink (UL) stream from the HMD to the rendering server also contained frequent and relatively small head-tracking information packets. Naturally, the Downlink (DL) stream also had regular video frame packet bursts. Fig. 2 is a visual representation of a short period of bidirectional VR streaming, showing

[2] https://riftcat.com/vridge

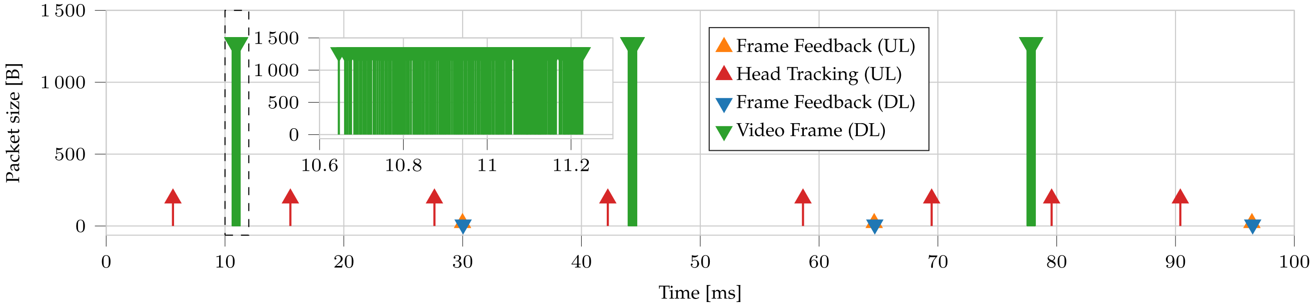


Fig. 2. Portion of traffic trace from *Virus Popper* (50 Mb/s, 30 FPS). In this trace, each video frame burst consists in about 130–140 individual fragments.

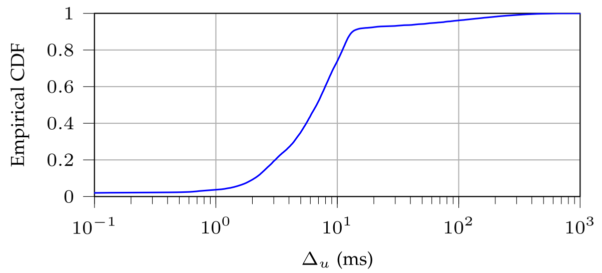


Fig. 3. Head tracking packet inter-arrival time.

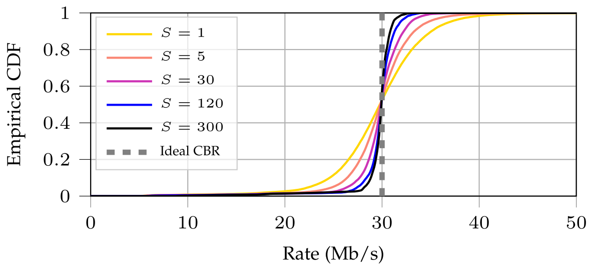


Fig. 4. Rate distribution for different MA window sizes S [number of frames].

the main data streams in both DL and UL. As the figure clearly shows, most of the traffic is concentrated in DL and consists of packet bursts encoding video frames. Video frame fragments were consistently 1320 B long in all acquired traces, with a data size (the UDP payload) of 1278 B.

First, we considered the head tracking packets in the UL, which were all 192 B long. The distribution of the inter-packet interval Δ_u is shown in Fig. 3: tracking packets are relatively frequent, with a median interval of about 7 ms, but the distribution has a long tail. This suggests that head-tracking packets are usually sent at regular intervals, but some are transmitted adaptively if there were significant headset movements that can affect the video rendering on the HMD. As we did not manage to decode the content of the tracking packets, a deeper analysis of their relation to head movements is left as future work.

By decoding the application protocol, we managed to identify frame boundaries and distinguish the video data frames from metadata and control information. We can then consider the size of individual frames in a video trace. We note that non-video packets have a low impact on the total streaming data rate. Considering this, as well as the strong dependence of metadata on the application setup, we decided to focus mostly on the video frame data, discarding all other packets from our analysis. Our results can then be applied to any VR application using the same encoder.

The encoder uses the H.264 *Periodic Intra-Refresh* compression scheme to reduce the variation between frame sizes, so we do not expect a multimodal distribution, as would be the case for a classical keyframe-based encoding. As we mentioned above, encoding VR traffic as CBR offers a significant advantage for the network optimization, because frames of constant size make it possible for NS schemes to provide a guaranteed latency without wasting resources.

However, CBR encoding is not perfect, and frames may still have variable size, although the average rate almost perfectly matches the required one. We can use a simple Moving Average (MA) filter with a rectangular window S to examine the behavior of the traffic on longer timescales, which is useful if resource allocation is performed at a slower pace. Naturally, allocating resources every S frames leads to a larger jitter between frames, but it can also improve the resource allocation efficiency, as size fluctuations tend to average out over multiple frames.

In order to measure this effect, we consider the *Virus Popper* trace, with a required rate $R = 30$ Mb/s and a $\varphi = 60$ FPS refresh rate. We only measure the video traffic, without packet headers and redundancy added by the application, which results in an average rate of 29.76 Mb/s. Fig. 4 shows the empirical Cumulative Distribution Function (CDF) of the rate, considering different values of the MA window sizes. If we consider each frame individually, there is a significant variation in the rate, which gradually reduces as we increase the window size.

However, even looking at longer time horizons, traffic is still far from the ideal CBR: Fig. 5 shows the overflow rate, i.e., the difference between the actual rate and the expected 30 Mb/s CBR rate, as a function of the MA window sizes. The plot shows the standard deviation, as well as the 95th and 99th percentile overflow rates. If our aim is to provide 99% reliability, we need to overprovision by more than 8 Mb/s (i.e., almost 30% of the CBR rate) even if we consider a timescale of 100 ms for resource allocation, i.e., 6 frames. Even averaging over periods of multiple seconds leads to worst-case rates almost 4 Mb/s higher than the target, probably because of highly dynamic content

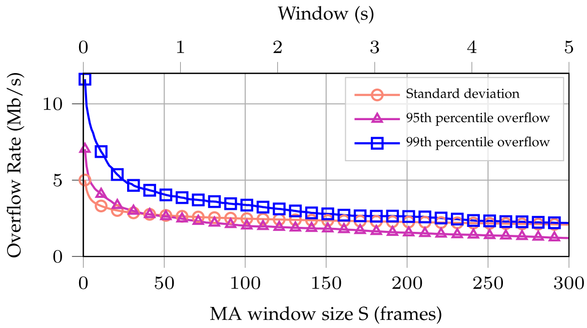


Fig. 5. Overflow rate for a target CBR of 30 Mb/s.

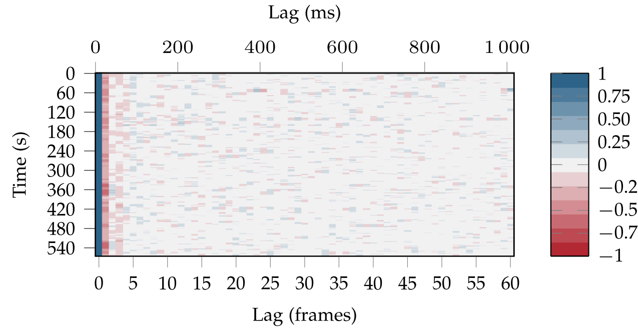


Fig. 7. Rolling windowed ΔF autocorrelation for *Virus Popper* (30 Mb/s, 60 FPS). The windows were 600 frames (10 s) long, with a time shift of 60 frames (1 s).

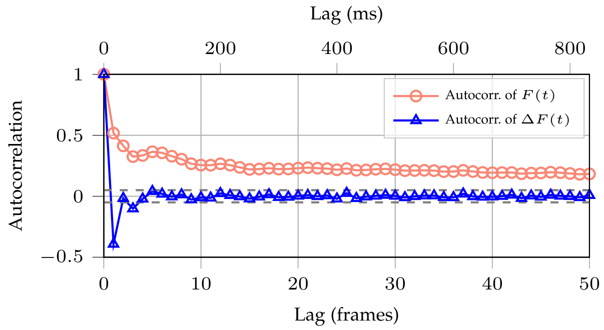


Fig. 6. Video frame size autocorrelation for *Virus Popper* (30 Mb/s, 60 FPS).

in the video. Interestingly, the overflow standard deviation is approximately constant if the MA window is longer than 50 frames, while the higher percentiles of the overflow continue to decay: this suggests that higher throughput periods tend to be shorter and more frequent, while there are longer periods of time with a bitrate below the average. Fig. 4 also hints at a skew in the distribution, as the left tail of the frame size empirical CDF is much longer.

Finally, we can analyze the autocorrelation of the frame size sequence $F(t)$, to identify patterns in how the sequence changes. Fig. 6 shows the autocorrelation of $F(t)$ and of $\Delta F(t) = F(t) - F(t-1)$. While $F(t)$ has a strong long-term autocorrelation, due to the constant component, the $\Delta F(t)$ sequence has a strong negative autocorrelation between one frame and the next, while almost all longer time differences fall within the ± 0.05 range. This means that the encoder tends to balance out fluctuations between one frame and the next, such that a frame that is bigger than the previous one tends to be followed by a smaller one again. We can check that this holds throughout the whole video by computing a rolling window autocorrelation, shown in Fig. 7 for $\Delta F(t)$. In this case, the plot clearly shows that there are no strong long-term correlations in any part of the video. The frame difference sequence has a noticeable and consistent autocorrelation only with lags 1, 3, and 5, confirming the result from Fig. 6. In the following, we will determine a prediction model for future frame sizes, exploiting the patterns we found in the frame size time series: the main symbols we use and their meaning are listed in Table I.

TABLE I
SYMBOLS USED IN THE PREDICTION MODEL DEFINITION

Symbol	Physical meaning	Unit
R	Average VR bitrate	Mb/s
φ	Frame rate	FPS (Hz)
$F(t)$	Size of frame t	kB
$\Delta F(t)$	Size variation between frames $t-1$ and t	kB
T	Prediction interval	Frames
$\bar{F}_T(t)$	Average size of the next T frames	kB
$\hat{F}_T(t)$	Predicted value of $\bar{F}_T(t)$	kB
N	Prediction model memory	Frames
τ	Prediction model delay	Frames
$\boldsymbol{\theta}$	Linear regression parameter vector	
p_s	Threshold probability	
δ	Robust linear regression parameter	kB
w	Residual prediction error	kB
K	Polynomial regression rank	
$\boldsymbol{\Psi}$	Polynomial regression parameter matrix	
$\hat{\mu}$	Residual error distribution mean	kB
$\hat{b}$	Residual error distribution shape parameter	kB
$\hat{\boldsymbol{\theta}}$	Normalized parameter vector	

A. *Frame Size Prediction*

Let us consider the average size of future frames in the time interval $[t, t+T)$, given by

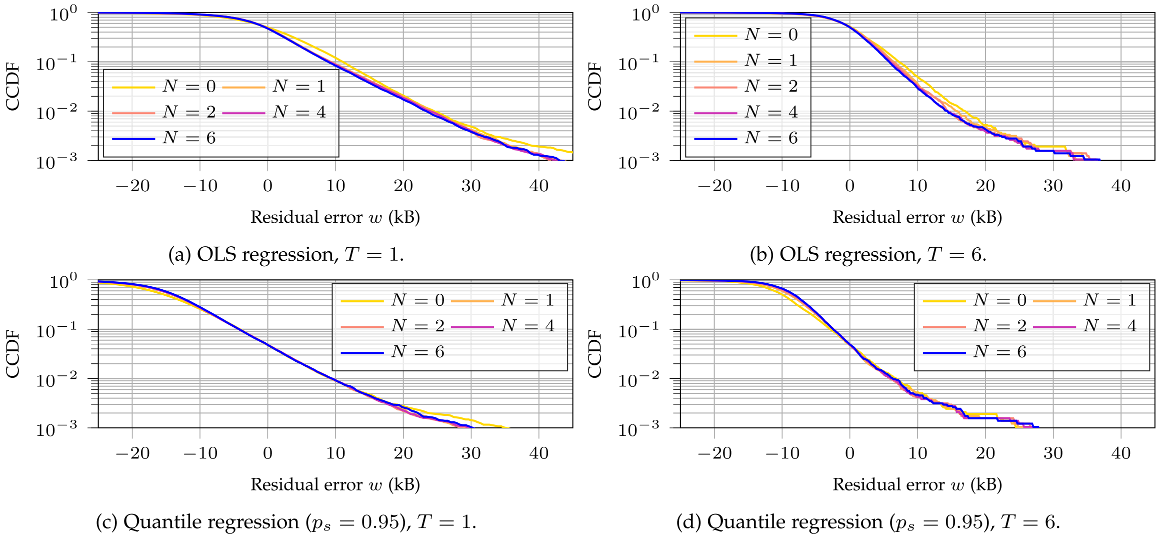
$$\overline{F_T}(t) = \frac{1}{T}\sum_{i=0}^{T-1} F(t+i). \tag{1}$$

We denote by $\hat{F}_T(t,\tau)$ an estimate of $\overline{F_T}(t+\tau)$, $\tau > 0$, i.e., considering a look-ahead of τ frames. We focus on linear predictors based on the last $N \geq 0$ samples, so that

$$\hat{F}_T(t,\tau) = \theta_0 + \sum_{j=1}^{N} \theta_j F(t-j+1), \tag{2}$$

where $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_N]$ is a weight vector, which determines the accuracy of the estimate. If $N = 0$, the estimate is just given by the parameter θ_0, and does not consider any past frames. The difference between actual and estimated value is captured by the error term $w(t,\tau,T) = \overline{F_T}(t+\tau) - \hat{F}_T(t,\tau)$, which will be denoted just as w in the following, for ease of writing. We can then consider two regression methods to determine the value of the parameter vector $\boldsymbol{\theta}$:

- *Ordinary Least Squares (OLS) linear regression:* least squares regression was independently developed by Gauss



(a) OLS regression, $T = 1$.

(b) OLS regression, $T = 6$.

(c) Quantile regression ($p_s = 0.95$), $T = 1$.

(d) Quantile regression ($p_s = 0.95$), $T = 6$.

Fig. 8. Complementary CDF of the error w with $\tau = 1$ and different values of N and T.

and Legendre in the 19th century [59], and is the most classic form of regression. In this case, the objective is to minimize the ℓ^2 norm of the sequence w. OLS regression can be useful in determining the average behavior of the underlying stochastic process, giving easily interpretable results on the quality of the prediction and the dynamics of the frame size over time;

- *Quantile regression [60]:* this technique estimates $\hat{F}_T(t, \tau)$ so that the probability that it is higher than the real value is not larger than p_s. This has obvious implications for network resource provisioning: as we are interested in providing enough resources to send a frame within the required latency with probability p_s, estimating the corresponding quantile might be the best way to get the required quality.

We also used *Robust linear regression* [61] to verify that the OLS prediction was not too sensitive to outliers. We considered a robust method using Huber's T norm instead of the ℓ^2 norm: the two norms have the same quadratic behavior if the error is smaller than a threshold δ, but Huber's T increases linearly for larger values. Setting the threshold to $\delta = \frac{\mathbb{E}[|F|]}{4}$, we found that the results matched exactly those of the OLS model, suggesting that outliers are not playing a relevant role in this case.

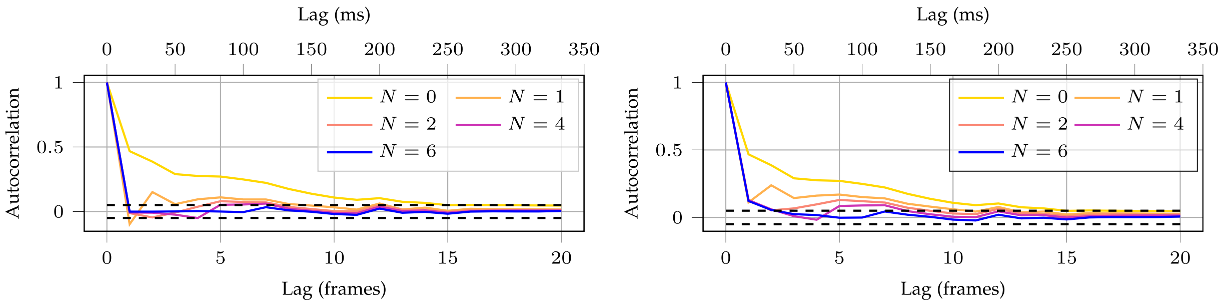
Fig. 8 shows the complementary CDF of the residual error w for the *Virus Popper* trace, considering a rate of 30 Mb/s and 60 FPS. We focus on this video trace as the standard example in the paper, but other traces, even at different bitrates and frame rates, exhibit a similar behavior. As we stated above, while the results from OLS are more immediate, quantile regression is useful when focusing on scheduling network resources for a VR stream, which requires a model of the tail of the frame size distribution to provide latency guarantees. In the following, we considered $\tau = 1$ and two different values of the averaging interval T.

The first thing we can notice from the figure is that the error distribution has a slightly different shape for the OLS and quantile regression models: indeed, our analysis of the model coefficients shows that the difference in the models is not simply caused by a shift in the value of the intercept θ_0, but the two models also have different coefficients for past samples. We can also notice that there is some benefit from having a longer memory, although increasing N yields diminishing returns in terms of increased accuracy. Finally, we can confirm that the reliable transmission of this VR content will require significant overprovisioning: for $T = 1$, the 95th percentile error of the OLS prediction is approximately 15 kB higher than the mean with any of the models, i.e., about 25% of the average frame size (which is 62.5 kB for this trace). In fact, this is close to the difference between the average predictions of the OLS and the quantile models.

This difference is about halved for $T = 6$, due to the fact that errors cancel each other out when computing the average over multiple frames. However, provisioning over multiple frames means that only the average amount of resources will be assigned to the stream, which will cause larger frames to have a higher latency, thus causing additional queuing delay to subsequent frames. Since the frame cannot be properly shown on screen until it is fully received, this translates to a higher jitter and reduces the QoE, making a lower value of T preferable.

Another fundamental component in evaluating the quality of a predictor is the autocorrelation of the residual error w: if the autocorrelation between subsequent samples of the residual error is high, the model did not capture some effect, usually due to an insufficient memory, i.e., too low a value of N. Fig. 9 shows the autocorrelation of w for different values of N: it is easy to see that models with $N < 4$, and particularly with $N = 0$ and $N = 1$, do not have enough memory to capture the frame size dynamics. This is more evident in quantile regression, which shows a higher autocorrelation for these models.

Finally, we can examine the effect of N and τ on the quality of the prediction by looking at Fig. 10, which shows the



(a) OLS regression.

(b) Quantile regression ($p_s = 0.95$).

Fig. 9. Autocorrelation of the residual error w for next-frame prediction ($T = 1$, $\tau = 1$) for different values of N.

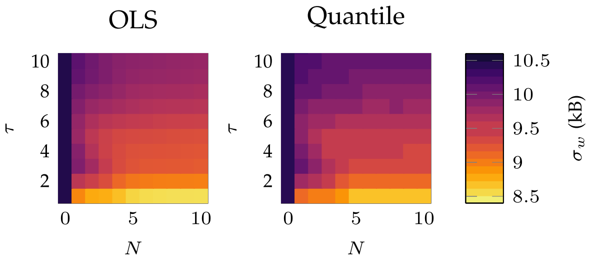


Fig. 10. Heatmap of the residual error standard deviation (measured in kB) as a function of N and τ, with $T = 1$.

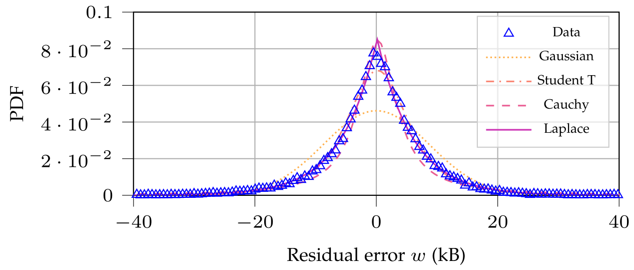


Fig. 11. Residual error after OLS prediction for *Virus Popper* (30 Mb/s, 60 FPS), with $T = 1$, $\tau = 1$, and $N = 6$.

standard deviation of the residual error w as a function of these two parameters with $T = 1$. The vertical axis of the colormap represents the feedback delay τ, while the horizontal axis maps to the memory N: we can easily see that the error increases with τ and decreases with N. The gains from increasing N when it is small (on the left side of the colormap) are significant, while the color changes imperceptibly between $N = 6$ and $N = 10$: the diminishing returns from increasing memory are consistent with our autocorrelation analysis, which showed that further increasing memory does not give any benefit in terms of prediction quality. Interestingly, the error is not a monotonically increasing function of τ for $N < 3$: this might be due to the autocorrelation we observed in the w sequence, as $N < 3$ is not sufficient to fully represent the state of the stochastic process, resulting in suboptimal predictions. In general, the error reduction from using $N = 6$ is around 20% with respect to the perfect CBR assumption.

B. Residual Error Characterization

We can then analyze the residual error w in more detail: as Fig. 11 shows, we attempted to fit the residual error on the frame size to various common bilateral distributions. The maximum likelihood fit was a Laplace(μ, b) distribution, whose Probability Density Function (PDF) is given by

$$p_w(x; \mu, b) = (2b)^{-1} e^{-\frac{|x-\mu|}{b}}, \tag{3}$$

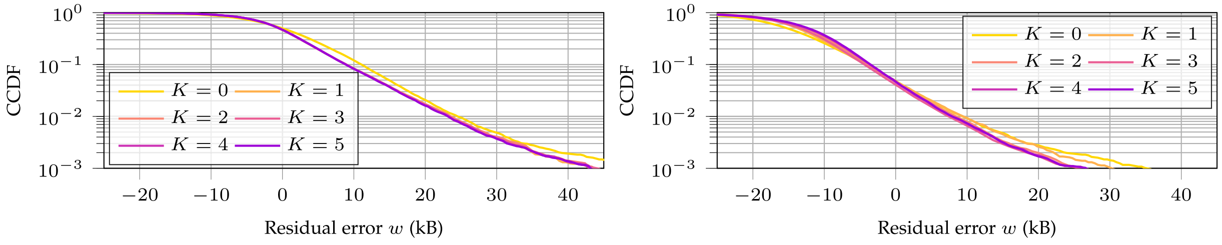
where μ and b are the location and shape parameters. The same result held for all other traces in the dataset, leading us to infer that this distribution depends on some inherent property of the encoder and the way it generates frames, instead of specific features in the video content.

If we consider the residual error of the OLS regression method, the best estimate of the parameter μ is $\hat{\mu} = 0$, as having a non zero-mean residual error would imply a bias in the OLS estimator. The maximum likelihood estimator of the shape parameter b is then given in [62] by

$$\hat{b} = \frac{1}{N} \sum_{i=1}^{N} |x_i - \hat{\mu}|. \tag{4}$$

The instantaneous value $\hat{b}_T(t, \tau)$ can then be determined from the model. As we are considering a regression model, we can simply perform an OLS regression on the magnitude of the residual error $|w|$ to find $\hat{b}_T(t, \tau)$. We can then represent the future frame size as a value $\hat{F}_T(t, \tau)$ given by the prediction plus a noise term w, whose distribution is Laplace$(\hat{F}_T(t, \tau), \hat{b}_T(t, \tau))$. Interestingly, if we adopt this model, we have the complete distribution of the frame size, making it extremely easy to derive the quantile values for any desired point and considerably reducing the computational impact with respect to multiple quantile regressions. The quantile function $P^{-1}(p_s|T, \tau, t)$ is then

$$P^{-1}(p_s|T, \tau, t) = \hat{F}_T(t, \tau) + \hat{b}_T(t, \tau) \log(\min(2p_s, 2 - 2p_s)). \tag{5}$$



(a) OLS regression.

(b) Quantile regression ($p_s = 0.95$).

Fig. 12. Complementary CDF of the error w ($T = 1$, $\tau = 1$, $N = 6$), with different polynomial regression ranks K.

C. Polynomial Regression

In order to further validate the model, and to ensure a better prediction, we extend our study to consider polynomial models: instead of a vector $\boldsymbol{\theta}$, a polynomial predictor of rank K with memory N is given by the constant parameter θ_0, along with a matrix $\boldsymbol{\Psi}$, so that

$$\hat{F}_T(t,\tau) = \theta_0 + \sum_{j=1}^{N}\sum_{k=1}^{K}\psi_{j,k}\left(F(t-j-\tau+1)\right)^k. \quad (6)$$

Fig. 12 shows the right tail of the residual error distribution as a function of the rank: we can note that the gain from considering polynomial regression with a rank higher than 1 is extremely limited for standard regression (i.e., predicting the size of the next frame with minimal Mean Square Error (MSE)), as shown in Fig. 12(a). On the other hand, Fig. 12(b) shows that the distribution of the residual error is skewed when performing polynomial quantile regression with $K > 1$, leading to a lower worst-case error but a very similar average error. While the performance of quadratic regression is slightly better with respect to linear quantile regression, we will use the linear model in the rest of this paper due to its simplicity, interpretability, and stability. As we discuss in the following, the linear model also provides more general results, with a smaller error when predicting frame size over different traces.

D. Model Generalization

In the above, we studied how well regression models can predict future frame sizes $\hat{F}_T(t,\tau)$, but we always found the parameter vector $\boldsymbol{\theta}$ based on the same video trace. In the following, we study how prediction models perform when the regression is performed over multiple traces, with different bitrates and types of content. Finding a predictor for each specific video content requires acquiring traces for each content and quality level, while generalizing the predictor would simplify the system.

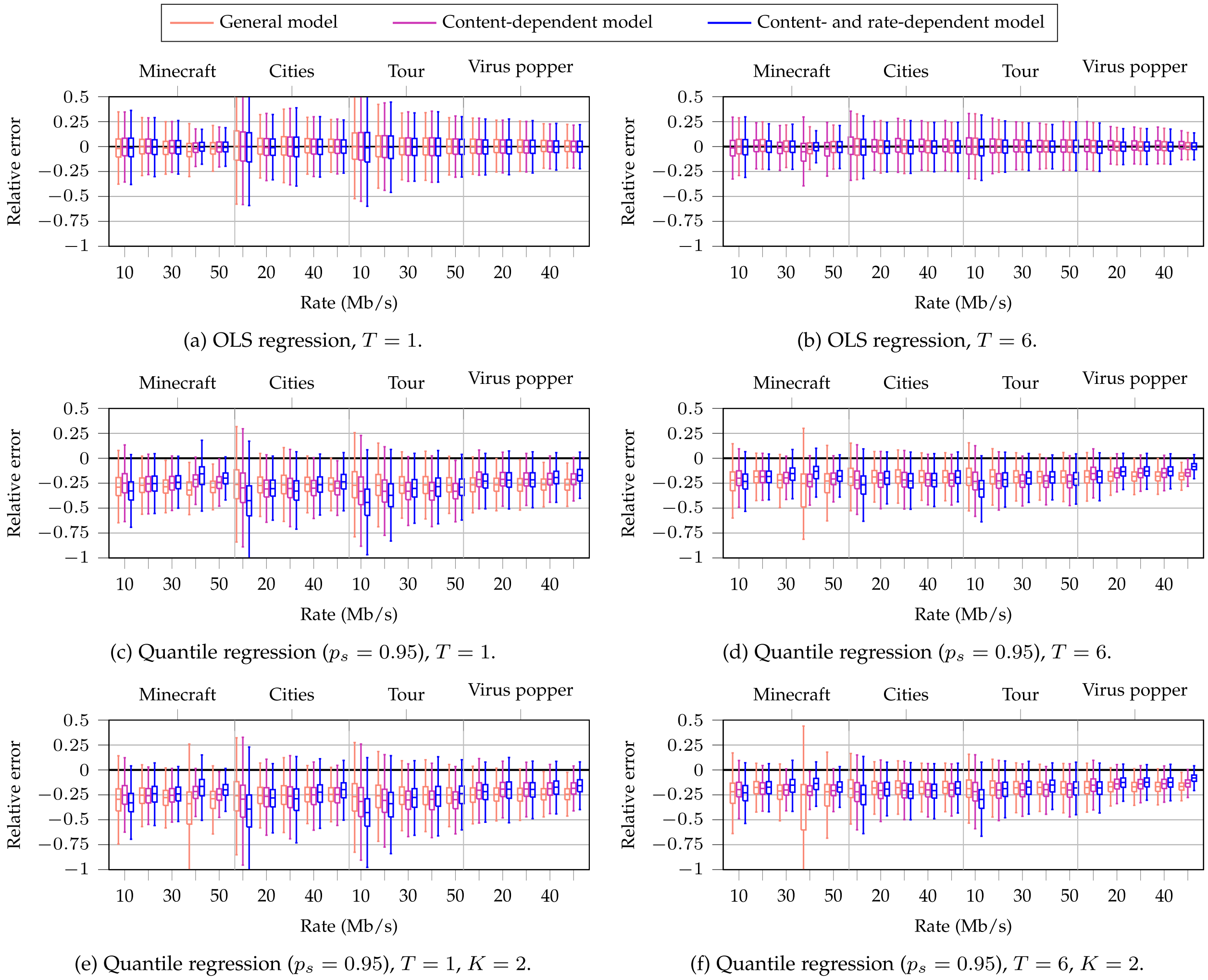
We consider $N = 6$ and $\tau = 1$, as we determined that $N = 6$ is sufficient to capture the dynamics of the model. In order to directly compare traces with different bitrates R and frame rates φ, we normalize the video traces by the expected frame size $\varphi^{-1}R$, obtaining a normalized parameter vector $\tilde{\boldsymbol{\theta}}$, which, given the linearity of our models, can be converted back to the original parameter vector in (1) as $\boldsymbol{\theta} = \frac{R\tilde{\boldsymbol{\theta}}}{\varphi}$. By normalizing our frame sizes, we can train and use our models on multiple traces with different values of R and φ. We then consider three generalized models:

1) A *general* model (GM), which computes $\boldsymbol{\theta}$ using the whole dataset, with different frame rates, bitrates, and video content types;
2) A *content-dependent* model (CM), which computes $\boldsymbol{\theta}$ using a single type of content (e.g., the *Virus Popper* game), but with different bitrates and frame rates;
3) A *content- and rate-dependent* model (CRM), which derives the parameter vector on a per-content, frame rate, and bitrate basis, i.e., a single trace.

Given that different values of R and φ may have different scales of errors which can be difficult to compare directly, in Fig. 13 we show the error normalized to the expected frame size R/φ. As the figure shows, the model can generalize quite well: the performance of CM is almost always similar to that obtained by CRM, making generalization across different bitrates and frame rates possible for the same video content. On the other hand, GM performs slightly worse, and has a large error in the Minecraft trace with $R = 40$ Mb/s: it is possible that this trace involves different dynamics in the content or head movements, leading to sharp differences even with other traces with the same type of content. On the other hand, GM has similar performance to CM and CRM with the OLS predictor, but shows a less consistent behavior for the quantile regressor. For example, the *Minecraft* trace with $R = 40$ Mb/s shows very different performance between the three models and different values of T. Furthermore, the *Virus Popper* trace seems to have a smaller tail, as GM is more conservative than the models based only on that video content.

If we consider the performance of the models across all videos, we can note that the general model introduces a slight negative bias of about 0.3% of the average size, but the gap between the 5th and 95th percentiles increases by about 2% with respect to the CRM model for each trace: this indicates that the derived model is highly general, as well as robust across different contents and rates.

As we can see, using the quantile model leads to a prediction between 25% and 40% higher than the average, skewing the error distribution. We also note that averaging over multiple frames can also significantly reduce the error across almost all traces. Finally, we can look at Fig. 13(e)–(f), which show the generalization performance of a polynomial quantile regressor with $K = 2$: we can note that the generalization performance



(a) OLS regression, $T = 1$.

(b) OLS regression, $T = 6$.

(c) Quantile regression ($p_s = 0.95$), $T = 1$.

(d) Quantile regression ($p_s = 0.95$), $T = 6$.

(e) Quantile regression ($p_s = 0.95$), $T = 1$, $K = 2$.

(f) Quantile regression ($p_s = 0.95$), $T = 6$, $K = 2$.

Fig. 13. Boxplot of the normalized residual error $\frac{\varphi w}{R}$ for different levels of generalization with $N = 6$ and $\tau = 1$. The traces are grouped by video content, and each group of boxplots shows the error at different bitrates for that video content.

is actually worse than in the linear model, providing further justification for the choice of a simpler regression model with fewer parameters.

V. Predictive Network Slicing

In this section, we consider an NS use case for the models we developed in Section IV. We assume that a number M of VR clients share the same RAN and need to minimize their resource consumption, while still delivering each VR frame of each client with strict latency constraints. Provisioning the time and frequency resources for VR is a critical component of Beyond 5 G networks, and guaranteeing limited latency while reducing the impact on other users is an important application of our model. Each client m then has a different bitrate R_m and, potentially, a different application, but we assume that the clients all share the same frame rate φ. Furthermore, each client m has a different spectral efficiency η_m, depending on its connection's Signal-to-Noise Ratio (SNR): users closer to the base station will have a stronger signal, and consequently, a higher transmission efficiency.

With a small loss of generality, we assume that clients are synchronized, i.e., frames are generated at the same time. The orchestration can be adapted relatively easily to the more general case, but the notation would be much more cumbersome, and we maintain this simplifying assumption for the sake of readability. We can then assume that the network slicing orchestrator is equipped with the general frame size distribution model from Section IV-D, and can estimate the frame size distribution for arbitrary values of T and τ for each client m. We consider an orchestrator that can make decisions on the resource allocation only at times $t = kS, k \in \mathbb{Z}$, i.e., every S frames or, conversely, every $\Delta t = \frac{S}{\varphi}$ ms. In the following, we consider queued bytes from earlier frames in the slicing as well. At time $t = kS$, we consider that the previous slice might have been unable to send all the data in time, leaving in the queue $q_m(t)$ bytes that have to be sent in the following frame intervals. The main parameters in the NS analysis are listed in Table II.

TABLE II
SYMBOLS USED IN THE NS DEFINITION

Symbol	Physical meaning	Unit
M	Number of VR clients	
R_m	Average bitrate for client m	Mb/s
φ	Frame rate	FPS (Hz)
η_m	Spectral efficiency for client m	b/s/Hz
S	Slicing decision interval	Frames
$q_m(t)$	Queued content after a frame interval	kB
Δ_u	Head tracking packet interval	ms
τ_m	Tracking delay	ms
τ_p	Propagation delay	ms
τ_f	Frame generation delay	ms
τ_r	Rendering delay	ms
$T(k)$	MTP latency for frame k	ms
$T_{\max}$	Maximum allowed MTP latency	ms
$B(k)$	Bandwidth required by frame k	MHz
T_{tx}	RAN transmission time for frame k	ms
$B_{\text{SCH}}^{(m)}(kS+\ell)$	Bandwidth allocated by scheme SCH to client m's frame $kS+\ell$	MHz

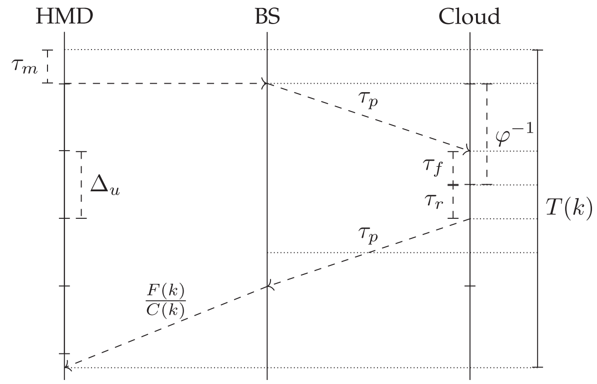


Fig. 14. Schematic of the components of the MTP latency.

A. Motion-to-Photon Latency

We can now analyze the MTP latency by dividing it into 6 components, which are shown in Fig. 14:

1) The movement of the user needs to be recorded and transmitted. For simplicity, we can assume head tracking packets to be transmitted at a constant interval Δ_u; in this case, the time between the motion and its transmission is $\tau_m \sim \mathcal{U}(0, \Delta_u)$.
2) The head tracking packet needs to be transmitted to the Cloud VR server. Considering that the uplink traffic is very light, and the packet is small, we can assume that it only incurs a constant propagation delay τ_p. We can also assume that Δ_u includes the uplink transmission time to the BS, simplifying the model. In general, the transmission time from the HMD to the BS should be extremely low: while the wireless link is often the bottleneck, cellular systems use orthogonal resources for the uplink and downlink, and short packets with commands and sensory readings incur negligible delays with respect to batches of full-sized data packets.
3) The head tracking data is received by the Cloud server, which then needs to produce a frame. The frame generation delay is $\tau_f \sim \mathcal{U}(0, \varphi^{-1})$.
4) The server needs to generate, render, and encode the frame. We denote this delay as τ_r, and assume that it is constant across short periods of time.
5) The frame is transmitted to the BS through a series of fiber optic links. As the capacity of fiber optic links is much higher than the RAN's, we can assume this to take only the propagation time τ_p.
6) The frame is transmitted from the BS to the HMD. This component depends on both the frame size and the downlink bandwidth allocated to its slice by the orchestrator.

If we set a maximum allowed MTP latency $T_{\max}$, we can then derive a condition on the minimum bandwidth $B(k)$ to be assigned to the nth customer in the kth interval of time

$$B(k) \geq \frac{F(k)}{\eta\left(T_{\max} - \tau_m - 2\tau_p - \tau_f - \tau_r\right)}. \tag{7}$$

where η is the spectral efficiency, known to the BS. However, τ_m and τ_f are random variables, so we can set a stricter condition that guarantees that the latency requirement is met in the worst case by substituting their maximum values, i.e., Δ_u and φ^{-1}, respectively. We hence obtain

$$B(k) \geq \frac{F(k)}{\eta\left(T_{\max} - \Delta_u - 2\tau_p - \varphi^{-1} - \tau_r\right)}. \tag{8}$$

For the sake of readability, we denote the maximum time allowed for the RAN transmission to fulfill the MTP latency requirement as T_{tx}, i.e.,

$$T_{\text{tx}} = T_{\max} - \Delta_u - 2\tau_p - \varphi^{-1} - \tau_r. \tag{9}$$

Finally, to ensure the stability of the queue at the BS, the average allocated bitrate, $\eta\mathbb{E}[B(k)]$, must be larger than the application's average bitrate, i.e.,

$$\eta\mathbb{E}[B(k)] > \varphi\mathbb{E}[F(k)]. \tag{10}$$

B. Slicing Schemes

We can then define four ways of allocating resources to the VR users:

1) *Individual FDMA (IF):* each individual VR user is allocated to a different slice that it can fully exploit, and each slice has a constant bandwidth over the next S frames. The bandwidth is then given by

$$B_{\text{IF}}^{(m)}(kS+\ell) = \frac{P_m^{-1}(p_s|S,1,kS) + \frac{q_m(t)}{S}}{\eta_m T_{\text{tx}}}, \tag{11}$$

with $\ell \in \{1,\ldots,S\}$. The scheme uses Frequency Division Multiple Access (FDMA), as the bandwidth B is constant over the whole slicing interval. In order to avoid instability, the $q_m(t)$ queued bits need to be considered in the slicing, but they are spread out over the S frames in the slicing period, so as to avoid excessive overprovisioning.
2) *Individual OFDMA (IO):* in this case, each user is still assigned to their own individual slice, but there is a finer-grained control over the assignment of bandwidth resources, allowing frame-by-frame control of the bandwidth assignment by using Orthogonal Frequency Division Multiple Access (OFDMA). In this case, the queued

bits can be handled in the first frame

$$B_{\text{IO}}^{(m)}(kS+1) = \frac{P_m^{-1}(p_s|1,1,kS) + q_m(t)}{\eta_m T_{\text{tx}}}. \tag{12}$$

In all subsequent frames, i.e., for $\ell \in \{2,\ldots,S\}$, the bandwidth assignment $B_{\text{IF}}^{(m)}(t)$ is then given by

$$B_{\text{IO}}^{(m)}(kS+\ell) = \frac{P_m^{-1}(p_s|1,\ell,kS)}{\eta_m T_{\text{tx}}}. \tag{13}$$

3) *Aggregated FDMA (AF):* while the two schemes described above give each user a slice of their own, this scheme performs FDMA, so it maintains a constant bandwidth throughout, but considers a single slice for the VR service, which is shared by all users. This allows users with larger than expected frames to exploit the bandwidth left unused by others with smaller than expected frames, but requires another, more fine-grained scheduler to divide the resources among users, which we will describe below. If we consider an oracle prediction, the required bandwidth $B^*(kS)$ to deliver all the generated data is given by

$$B^*(kS) = \sum_{m=1}^{M} \frac{\frac{q_m(t)}{S} + \sum_{\ell=1}^{S} F^{(m)}(kS+\ell)}{\eta_m T_{\text{tx}}}. \tag{14}$$

We can express the required bandwidth as the sum of two components, $B_q^*(kS)$ and $B_f^*(kS)$

$$B_q^*(kS) = \sum_{m=1}^{M} \frac{q_m(t)}{S\eta_m T_{\text{tx}}}; \tag{15}$$

$$B_f^*(kS) = \sum_{m=1}^{M} \sum_{\ell=1}^{S} \frac{F^{(m)}(kS+\ell)}{\eta_m T_{\text{tx}}}. \tag{16}$$

While $B_q^*(kS)$ is a deterministic, known value, as it only depends on the amount of queued bytes for each user, the bandwidth $B_f^*(kS)$ required to transmit future frames is unknown, as the size of these frames is stochastic. The distribution of $B_f^*(kS)$ is given by the convolution of MS Laplace distributions, and the details of its computation are given in the Appendix, available online. If we denote the quantile function of this distribution as $P_{1,\ldots,M}^{-1}(p_s|S,1,t)$, we get, with $\ell \in \{1,\ldots,S\}$

$$B_{\text{AF}}(kS+\ell) = \frac{P_{1,\ldots,M}^{-1}(p_s|S,1,kS) + \sum_{m=1}^{M} \frac{q_m(t)}{S}}{\eta_m T_{\text{tx}}}. \tag{17}$$

We assume that users are synchronized, so that frames arrive approximately at the same time: this results in a need-based scheduler delivering the frames from all users approximately at the same time, allocating more bandwidth to users with a larger frame (or a lower spectral efficiency). This is a slight simplification, but we can easily adapt the mechanism to the asynchronous case with a limited loss of performance. The choice of a need-based scheduler leaves the decision of setting user rates to flow admission, serving users equitably once they access the system.

TABLE III
BASIC SCENARIO PARAMETERS

Parameter	Physical meaning	Value
$T_{\max}$	Maximum MTP latency	50 ms
Δ_u	Head tracking interval	7 ms
τ_p	Propagation time	5 ms
τ_r	Rendering time	5 ms
p_s	Prediction quantile	0.95
S	Slicing decision interval	6 frames
N	Predictive model memory	6 frames
τ	Predictive model delay	1 frame
R	Average bitrate	30 Mb/s
φ	Frame rate	60 FPS
Content	VR application	Virus Popper

4) *Aggregated OFDMA (AO):* as we did for the individual slicing, we can also create aggregated slices with a finer-grained control of the bandwidth allocation. In the first frame, the queues need to be flushed before new data can be transmitted

$$B_{\text{AF}}(kS+1) = \frac{P_{1,\ldots,M}^{-1}(p_s|1,1,kS) + \sum_{m=1}^{M} q_m(t)}{\eta_m T_{\text{tx}}}. \tag{18}$$

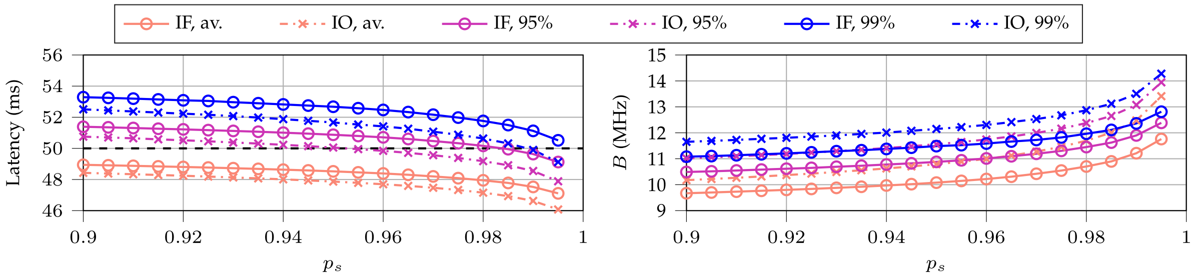
For $\ell \in \{2,\ldots,S\}$, we then have

$$B_{\text{AO}}(kS+\ell) = \frac{P_{1,\ldots,M}^{-1}(p_s|1,\ell,kS)}{\eta_m T_{\text{tx}}}. \tag{19}$$

Naturally, if there is only one user, i.e., $M = 1$, the AF and AO slicing schemes are the same, as are the IF and IO, respectively. In the same way, the FDMA and OFDMA slicing schemes are equivalent if $S = 1$, as the allocation is performed over the shortest possible unit of time, i.e., a single frame period. All slicing schemes ensure the stability of the queue by considering $q_m(t)$ in the bandwidth allocation, inherently ensuring that the condition in (10) is met. The calculation of the aggregated traffic distribution is given in the Appendix, available online.

VI. SIMULATION RESULTS

In order to evaluate the performance of predictive slicing schemes, we run a Monte Carlo simulation based on a simple system: we set fixed values for τ_p and τ_r, and considered a simple point to point channel with a constant capacity. The simulations were run through a Python script, which is available with the analysis code, and are meant to highlight the trade-off between the two fundamental Key Performance Indicators (KPIs) of the system: the MTP latency and the bandwidth B reserved to the VR users. Ideally, a system should be able to maintain the MTP latency below the required threshold while limiting the required bandwidth. The main parameters of the scenario are listed in Table III, and will be used in all simulations, unless stated otherwise. We chose $T_{\max} = 50$ ms, which is consistent with the relevant literature, though looser than in the IEEE standard [22]: as the application we used for measuring has $\Delta_u = 7$ ms (on average) and $\varphi = 60$ FPS, even an instantaneous transmission would incur an MTP latency over 20 ms in the worst case. The stricter deadline set by the IEEE standard is then impossible to meet with the considered application, and we chose a looser but still realistic deadline, leaving the fulfillment



(a) Latency.

(b) Assigned bandwidth.

Fig. 15. Average and worst-case percentiles of the latency and assigned bandwidth of a single user as a function of the quantile p_s, with $S = 6$.

of the more demanding one to future work with more powerful XR applications. The simulation setup is simplified, as it only involves a simple channel without any mobility or wireless fading, but it can highlight the main features of predictive NS. More complex scenarios, which could include channel prediction into the system, will be considered as future work.

We also considered $\tau_p = 5$ ms and $\tau_r = 5$ ms, considering a powerful Cloud VR server located relatively close to the user. The final parameters are close to the 3GPP recommendation [38], which specifies a rate $R = 30$ Mb/s and $\varphi = 60$ FPS, although our DL latency budget is slightly looser, as the BS has 11.3 ms to stream each frame, while 3GPP specifies 10 ms as the target.

A. Single User

We can now examine the simulation results for a single user. Fig. 16 shows the schemes' latency and assigned bandwidth as a function of S, setting $p_s = 0.95$ and $N = 6$: we can note that IO manages to maintain a constant latency, with a 95th percentile latency of 51 ms, while the latency for IF tends to increase with S. This is due to the effect mentioned above, as IO can assign each frame enough resources to satisfy the latency bound even in the worst-case outcome, while IF must rely on a rougher prediction. However, this comes at a cost: IO needs to allocate more bandwidth resources to maintain the latency as S grows, while IO can allow some frames to arrive with a higher latency, compensating the frame size variation while using fewer resources. The performance of static slicing is not shown here, as it would require expanding the boundaries of the plot: a fixed allocation close to the bandwidth that IF and IO use leads to a significantly higher average latency, with peaks of hundreds of milliseconds, highlighting the need for predictive slicing schemes.

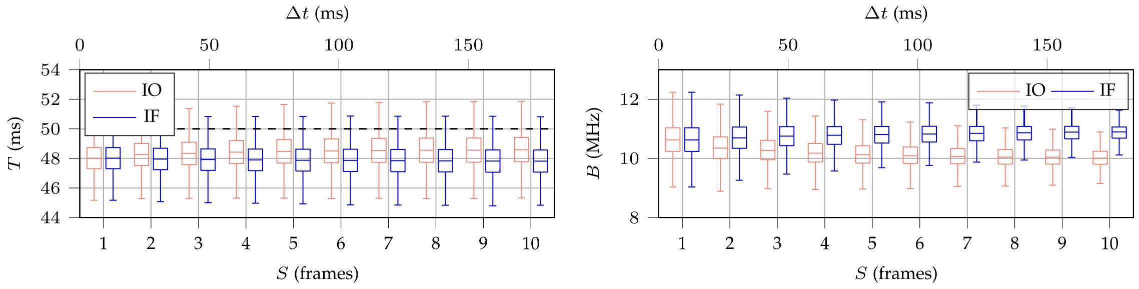
Both models are realistic, as they work under different assumptions: in the first case, the resources that are allocated for each frame need to be over both time and frequency, while the second case gives the slice a constant bandwidth over the slicing interval, which is the most common slicing model in the literature. Naturally, the choice between the two models depends not only on the desired point in the trade-off between QoS and resource efficiency, but also on the capabilities of the underlying system: state-of-the-art slicing frameworks often consider a period $\Delta t = 100$ ms, which would correspond to $S = 6$ frames, and the granularity of the slicing over time and frequency will dictate whether IO is even an option.

We can also consider the performance of the schemes as a function of p_s, which allows the schemes to move along the trade-off between bandwidth and latency: higher values correspond to more extreme percentiles on the tail of the frame size distribution, causing the NS algorithms to assign more bandwidth to the VR flow. Fig. 15 shows the slicing performance as a function of the value of p_s. We can notice that the closer p_s gets to 1, the larger the effect of small changes in the parameter becomes, as we approach the right tail of the estimated distribution. Additionally, IO maintains an advantage over IF in terms of latency, which is around 1 ms for all percentiles, but on the other hand, it also requires between 500 kHz and 1 MHz more at any given percentile. The trade-off between latency and bandwidth is crucial, and by setting the appropriate value of p_s, we can adapt the priorities of the schemes. Interestingly, setting $p_s = 0.985$ with IF results in similar performance to IO with $p_s = 0.95$, with an only slightly higher bandwidth and the same latency. This shows that a properly tuned IF can perform almost as well as IO, without the additional requirements for finer-grained slicing: in this case, IF is a more attractive scheme for implementation, as it fits better with the assumptions of most RAN NS solution. The only downside is that choosing the correct value of p_s to compensate for the slicing scheme's optimism is not simple, particularly in more complex network scenarios, while it is relatively straightforward for IO.

B. Multiple Users

We can now consider a more complex scenario, in which multiple VR users are served by the same BS. We will first analyze the impact of the number of users M on the system, considering the case in which all users have the same spectral efficiency $\eta = 5$ b/s/Hz, and stream the same content (i.e., the *Virus popper* trace at 60 FPS and 30 Mb/s), although starting with a random offset. Furthermore, as we stated in Section V-B, we assume that all VR users are synchronized, i.e., frames from all users come at the same time: this is a slight simplification, which does not have a significant effect on performance.

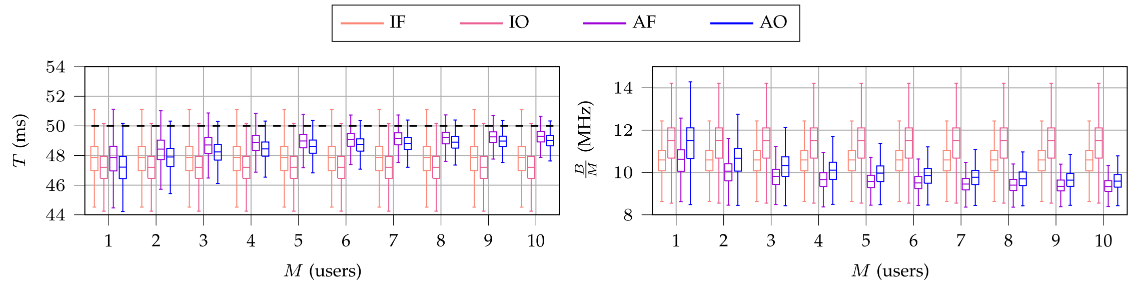
Fig. 17 shows the latency and assigned bandwidth for the scenario: as for the single user case, static slicing is not shown,



(a) Latency.

(b) Assigned bandwidth.

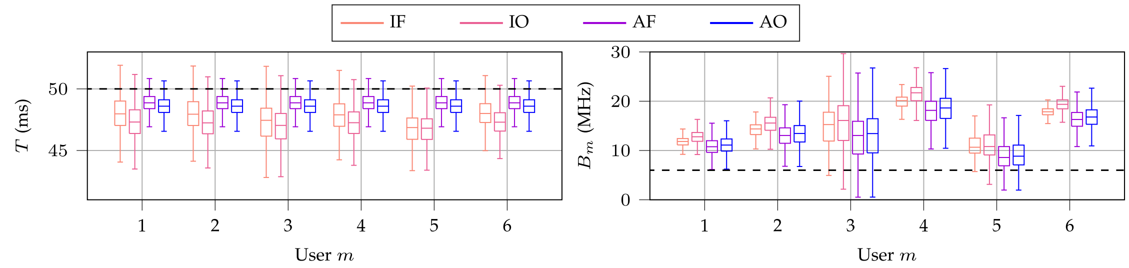
Fig. 16. Boxplot of slicing performance for IF and IO for a single user.



(a) Latency.

(b) Assigned bandwidth per user.

Fig. 17. Boxplot of the latency and per-user bandwidth as a function of the number of users.



(a) Latency.

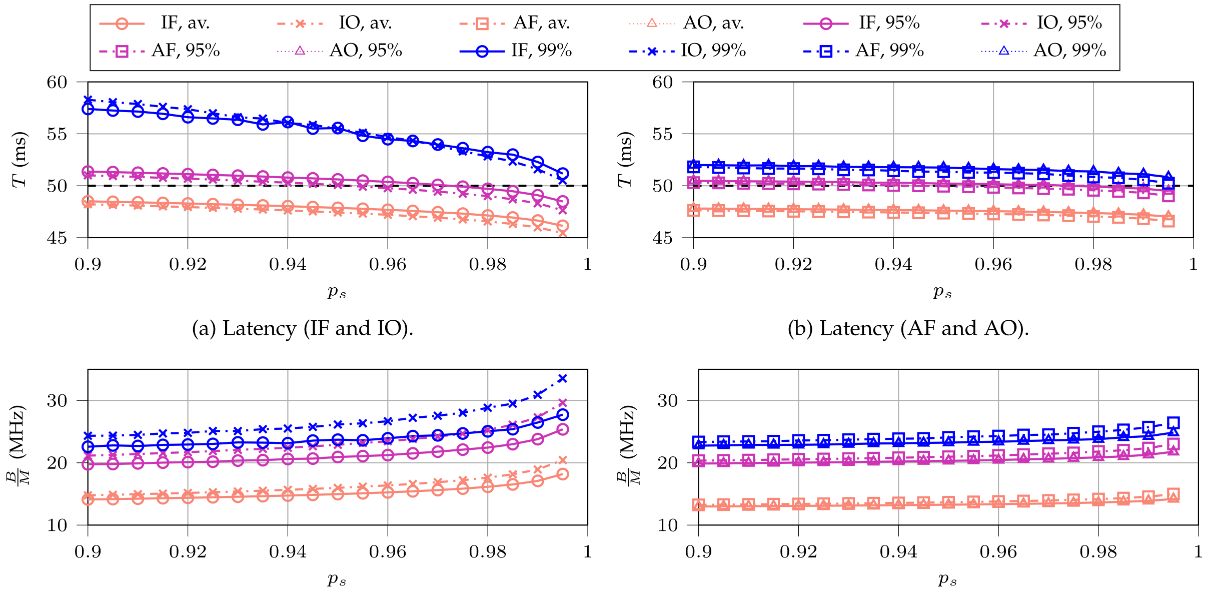
(b) Assigned bandwidth.

Fig. 18. Boxplot of the latency and bandwidth for each user in the scenario from Table IV.

as the latency for similar bandwidth allocations can explode, with peaks close to 1 s. In general, IF and IO maintain a lower average latency, as shown by Fig. 17(a), but their aggregated versions have the same maximum latency violation probabilities. The same pattern that was present in the single user simulations, with the FDMA slicing schemes having a higher latency but using less bandwidth than the OFDMA ones, is present here. However, there is an interesting pattern in Fig. 17(b): while the IF and IO schemes have the same per-user bandwidth performance regardless of the number of users, AF and AO gradually use less bandwidth for each user as M increases. This is a significant advantage with respect to individual slicing, which is compounded by the lower jitter: it is caused by the aggregated schemes compensating for larger frames by exploiting the law of large numbers. Aggregating multiple users results in a distribution with a relatively shorter tail, as the 95th percentile of the sum of M random variables is smaller than the sum of the 95th percentiles of their M distributions. This effect has a dual benefit, as it simplifies the NS algorithm, grouping all VR users in the same slice and using simpler scheduling mechanisms to assign resources inside the slice.

TABLE IV
MULTI-USER SCENARIO PARAMETERS

User m	VR content	R_m (Mb/s)	η_m (b/s/Hz)	$\bar{B}_m$ (MHz)
1	Virus Popper	10	1.5	6.67
2	Cities	20	2.5	8
3	Minecraft	30	3	10
4	Tour	40	3.5	11.43
5	Minecraft	50	5.5	9.09
6	Virus Popper	40	4	10



(a) Latency (IF and IO).

(b) Latency (AF and AO).

(c) Assigned bandwidth (IF and IO).

(d) Assigned bandwidth (AF and AO).

Fig. 19. Average and worst-case percentiles of the latency and assigned bandwidth as a function of the quantile p_s in the multi-user scenario defined by the parameters in Table IV.

We can also analyze the results for each user in a more challenging scenario, whose detailed parameters are given in Table IV: each user has a different video bitrate R_m and a different spectral efficiency, but they have a common frame rate φ and are synchronized.

Fig. 18 shows the latency and assigned bandwidth for the 6 users, presenting some interesting patterns. As Fig. 18(a) shows, the latency for all users is the same when using the aggregated slicing schemes, as resources are assigned with a fair scheduler, while individual slicing can cause some additional variability for users with a higher frame size variability. However, in general all slicing schemes try to maintain a latency close to the bound, avoiding the waste of bandwidth resources while respecting the latency requirements. On the other hand, the bandwidth assigned to each user, shown in Fig. 18(b), is similar for all schemes, with the aggregated ones having a slightly lower resource utilization. This is due to the fact that the average required bandwidth $\bar{B}_m = \frac{R_m}{\eta_m}$, whose value for each user is given in Table IV, is the main factor in assigning bandwidth resources, both with individual and aggregated schemes: we can easily see that users 1, 2, and 5, who have the lowest values of $\bar{B}_m$, are also assigned the least bandwidth by the slicing schemes. Interestingly, user 5 is actually the one with the lowest bandwidth, although users 1 and 2 have lower values of $\bar{B}_m$: this is due to the higher relative variations in the video traces at lower bitrates, as can be easily seen from Fig. 13, so that in order to meet the $p_s = 0.95$ requirement, the slicing schemes have to overprovision more for those users.

C. Performance Considerations

In order to draw some design insights from the results in the previous subsections, we consider the scenario given in Table IV and look at the performance as a function of p_s, as we did for the single-user case. Fig. 19 shows the performance of the individual (on the left) and aggregated (on the right) slicing schemes in the multi-user scenario defined in the previous section: it is easy to see that aggregated schemes are superior, as they both maintain an overall latency closer to the bound at all percentiles, with a much lower jitter, and use significantly fewer resources. We can note that the fine-grained allocation of resources that OFDMA allows is not actually necessary, as the performance difference between it and FDMA is negligible: this is because the prediction errors are effectively compensated by other users or frames, significantly simplifying the scheduling requirements.

We can further analyze the performance difference between the schemes by plotting their Pareto curves. Pareto curves are useful to show two-dimensional performance metrics which need to be traded against one another: the curve includes all points at the edge of the achievable performance region, i.e., points for which it is impossible to improve one metric without making the other worse. In our case, the two metrics are the MTP latency and the bandwidth: ideally, we would like both to be as low as possible, and the Pareto curve is another way of showing the trade-off we discussed above.

If we define the performance of a slicing scheme g in terms of latency and bandwidth as $q_g(p_s) = (T, B)$, we can say that p_s *dominates* p'_s, and we write $p_s \succ p'_s$ where p_s has a better performance for both metrics, i.e.,

$$T(g, p_s) < T(g, p'_s) \wedge B(g, p_s) < B(g, p'_s). \quad (20)$$

We can then define the Pareto curve $\mathcal{P}_g$ as the set of points that are not dominated by any other point

$$\mathcal{P}_g \triangleq \{q_g(p_s), \forall p_s \mid \nexists p'_s : p'_s \succ p_s\}. \quad (21)$$

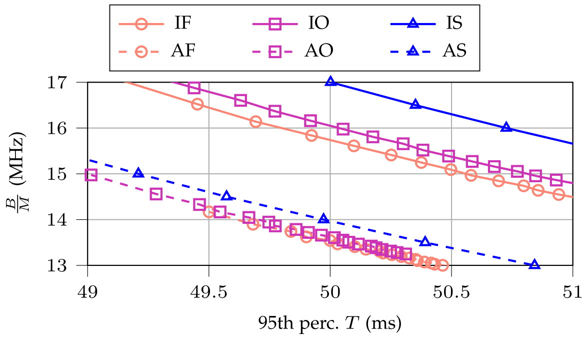


Fig. 20. Pareto curve of slicing scheme performance.

Fig. 20 shows the Pareto curves for the four schemes, considering $p_s \in [0.9, 0.995]$. The two plots show the performance in terms of the average assigned bandwidth per user and the 95th and 99th percentiles of latency, and confirm our earlier analysis: the aggregated schemes can significantly outperform the individual ones, with a bandwidth reduction of more than 10% to obtain the same latency performance at the 95th percentile and more than 20% at the 99th percentile. We can also note that, while the difference between IF and IO is relatively small for the 95th percentile, it grows for the 99th percentile, as taking the worst case highlights the limits of the FDMA approach. On the other hand, the difference between AF and AO is negligible. The figure also shows a comparison with the Individual Static (IS) and Aggregated Static (AS) allocation schemes, which give a constant bandwidth to each slice and represent the state of the art in NS. We can note two things from the comparison: first, assigning each user to an individual slice is wasteful even with respect to static allocation, and second, the AF and AO schemes reduce the bandwidth consumption per user by about 5% with respect to the AS scheme, showing the effectiveness of predictive slicing. This gain is even more noticeable for individual slices, and cases with 1 or 2 simultaneous clients will show a bandwidth reduction of about 10% for the same latency reduction.

In addition to significantly improving the performance, aggregated schemes can also simplify the NS mechanism, as they require fewer decisions by grouping VR users in a single slice. This fact, along with the negligible performance gap with respect to AO, makes AF the most attractive algorithm for a real implementation in 5 G and Beyond networks. We remark again that the need for predictive slicing is dire, as static slicing schemes cannot provide reliable MTP latency performance without significant resource overprovisioning. All these considerations are even more significant for VBR VR content, as the variations with that encoder setting are at least an order of magnitude bigger.

VII. Conclusion

This work aims at closing a gap in the literature on traffic source modeling: there are several analyses for passive streaming, both 2D and in immersive setups with Head Mounted Devices (HMDs), and some for live gaming traffic in 2D, but none for interactive VR with strict latency requirements and *quasi*-CBR encoding. We analyzed live captures from a setup we devised, publishing both the dataset and the code for the analysis, and presented the performance of two regression models. The first part of our discussion analyzes the prediction models, determining the necessary memory in the linear regression, the residual distribution, and the correctness of the linear model. The prediction models are simple and flexible, as they generalize extremely well across different traces and bitrate settings: this means that a shared pre-trained model can be used with good performance across different video contents and bitrates.

We then showed a simple Network Slicing (NS) scenario, which highlights the importance of the trade-off between resource efficiency and QoE. This is a first step towards fully designing an NS system able to satisfy the stringent QoS requirements of XR applications also in critical scenarios, e.g., in industrial settings, in which the consequences of network failures are not only discomfort and nausea for the user, but also significant delays in production and even safety hazards. The results we obtained show a significant trade-off between resource efficiency and MTP latency guarantees, which can be improved significantly if multiple VR users are put together in the same slice, sharing Radio Access Network (RAN) resources using a fair scheduler.

The results that we obtained are specific to the H.264 *quasi*-CBR mode, but the basic experimental and analytical methodology would be the same for other encoders, or even VBR traffic. In the latter case, we expect the variability of frame size, and the error in any prediction, to be higher by up to an order of magnitude, making predictive slicing without any application-level inputs extremely hard.

There are several additional analyses and opportunities for future work, that can be divided in two main directions. The first potential avenue of research is a wider characterization, with different encoding parameters and even different encoders, and considering different applications, going beyond simple VR games to include the industrial and commercial use cases we mentioned above, and a wider set of subjects. The traces should also integrate a record of the head movements of the users, as they correspond to shifts in the point of view of the VR headset and are expected to be strongly correlated with frame size changes. In order to consider the complex interactions between user movements, video content, and the generated traffic, more advanced predictors such as neural networks could be helpful, but would incur the risk of overfitting the model and the need for a wider dataset.

The other challenge is to actually design slicing schemes and scheduling algorithms able to take into account the nature of the traffic and accommodate it, efficiently exploiting the prediction and adapting to the peculiarities of different communication technologies or even multiple independent links. The use of packet-level coding to protect the stream from link failures and deep fading events can be a promising avenue to design a solid framework to support XR in mission-critical scenarios. The study of these techniques at all levels of the communication stack, simulating connection impairments in repeatable conditions through a full-stack network simulator, is our first priority in the ongoing work on this subject.

References

[1] Empowering consumer-focused immersive VR and AR experiences with mobile broadband, Huawei Technologies Co., White Paper, 2016. Accessed: Mar. 15, 2023. [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/mbb/huawei_whitepaper_vr_ar_final.pdf?la=en

[2] AR insight and application practice, Huawei Technologies Co., White Paper, 2021. Accessed: Mar. 15, 2023. [Online]. Available: https://carrier.huawei.com/ /media/CNBGV2/download/bws2021/ar-insight-and-application-practice-white-paper-en.pdf

[3] The future of work: How VR is changing the way we collaborate and get things done, Oculus Blog, White paper, Sep. 2020. Accessed: Mar. 15, 2023. [Online]. Available: https://www.oculus.com/blog/connect-future-of-work-vr/

[4] The mobile future of eXtended reality (XR), Qualcomm Technologies, Presentation, Nov. 2020. Accessed: Mar. 15, 2023. [Online]. Available: https://www.qualcomm.com/media/documents/files/the-mobile-future-of-extended-reality-xr.pdf

[5] Extended reality (XR) in 5G, 3GPP, Technical Report (TR) 26.928, Dec. 2020. [Online]. Available: https://www.3gpp.org/DynaReport/26928.htm

[6] ITU-T, "Requirements for mobile edge computing enabled content delivery networks," ITU, Report F.743.10, Nov. 2019. [Online]. Available: https://www.itu.int/rec/T-REC-F.743.10

[7] H. G. Kim, W. J. Baddar, H.-T. Lim, H. Jeong, and Y. M. Ro, "Measurement of exceptional motion in VR video contents for VR sickness assessment using deep convolutional autoencoder," in *Proc. 23rd ACM Symp. Virtual Reality Softw. Technol.*, 2017, Art. no. 36.

[8] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Comput. Netw.*, vol. 167, Feb. 2020, Art. no. 106984.

[9] 5G; Management and orchestration; Concepts, use cases and requirements, 3GPP, Technical Specification (TS) 28.530, Oct. 2022. [Online]. Available: https://www.3gpp.org/DynaReport/28530.htm

[10] Y. Wu, H.-N. Dai, H. Wang, Z. Xiong, and S. Guo, "A survey of intelligent network slicing management for industrial IoT: Integrated approaches for smart transportation, smart energy, and smart factory," *IEEE Commun. Surv. Tuts.*, vol. 24, no. 2, pp. 1175–1211, Second Quarter 2022.

[11] S. E. Elayoubi, S. B. Jemaa, Z. Altman, and A. Galindo-Serrano, "5G RAN slicing for verticals: Enablers and challenges," *IEEE Commun. Mag.*, vol. 57, no. 1, pp. 28–34, Jan. 2019.

[12] C. An and T. Q. Nguyen, "Iterative rate-distortion optimization of H.264 with constant bit rate constraint," *IEEE Trans. Image Process.*, vol. 17, no. 9, pp. 1605–1615, Sep. 2008.

[13] I. Shin, J. Yoo, and J. Hong, "Distortion estimation in CBR channel and its application for H.264 rate control," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2008, pp. 689–692.

[14] S. Zhao, H. Abou-zeid, R. Atawia, Y. S. K. Manjunath, A. B. Sediq, and X.-P. Zhang, "Virtual reality gaming on the Cloud: A reality check," in *Proc. IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.

[15] F. Aumont, F. Humbert, C. Neumann, C. Salmon-Legagneur, and C. Taibi, "Dissecting cloud game streaming platforms regarding the impacts of video encoding and networking constraints on QoE," in *Proc. Workshop Game Syst.*, 2021, pp. 13–19.

[16] Cloud gaming and 5G–Realizing the opportunity, Nokia Bell Labs Consulting, White Paper, 2020. Accessed: Mar. 15, 2023. [Online]. Available: https://onestore.nokia.com/asset/207843

[17] Preparing for a Cloud AR/VR future, Huawei Technologies Co., White Paper, 2017. Accessed: Mar. 15, 2023. [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/x-lab/cloud_vr_ar_white_paper_en.pdf?la=en

[18] M. Lecci, A. Zanella, and M. Zorzi, "An ns-3 implementation of a bursty traffic framework for virtual reality sources," in *Proc. Workshop ns-3*, 2021, pp. 73–80.

[19] M. Lecci, M. Drago, A. Zanella, and M. Zorzi, "An open framework for analyzing and modeling XR network traffic," *IEEE Access*, vol. 9, pp. 129782–129795, 2021.

[20] M. Lecci, F. Chiariotti, M. Drago, A. Zanella, and M. Zorzi, "Temporal characterization of XR traffic with application to predictive network slicing," in *Proc. IEEE 23rd Int. Symp. World Wireless Mobile Multimedia Netw.*, 2022, pp. 406–415.

[21] J. Latta and D. Oberg, "A conceptual virtual reality model," *IEEE Comput. Graph. Appl.*, vol. 14, no. 1, pp. 23–29, Jan. 1994.

[22] IEEE SA, "IEEE standard for head-mounted display (HMD)-based virtual reality(VR) sickness reduction technology," IEEE, Standard 3079–2020, Sep. 2020, doi: 10.1109/IEEESTD.2021.9416950.

[23] M.-W. Seo, S.-W. Choi, S.-L. Lee, E.-Y. Oh, J.-S. Baek, and S.-J. Kang, "Photosensor-based latency measurement system for head-mounted displays," *Sensors*, vol. 17, no. 5, May 2017, Art. no. 1112.

[24] A. Blate et al., "Implementation and evaluation of a 50kHz, 28μs motion-to-pose latency head tracking instrument," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 5, pp. 1970–1980, May 2019.

[25] J.-P. Stauffert, F. Niebling, and M. E. Latoschik, "Simultaneous run-time measurement of motion-to-photon latency and latency jitter," in *Proc. IEEE Conf. Virtual Reality 3D User Interfaces*, 2020, pp. 636–644.

[26] J. Zhao, R. S. Allison, M. Vinnikov, and S. Jennings, "Estimating the motion-to-photon latency in head mounted displays," in *Proc. IEEE Virtual Reality Conf.*, 2017, pp. 313–314.

[27] S.-W. Choi, S. Lee, M.-W. Seo, and S.-J. Kang, "Time sequential motion-to-photon latency measurement system for virtual reality head-mounted displays," *Electronics*, vol. 7, no. 9, Sep. 2018, Art. no. 171.

[28] S. Tanwir and H. Perros, "A survey of VBR video traffic models," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 4, pp. 1778–1802, Fourth Quarter 2013.

[29] Y. Chen, K. Wu, and Q. Zhang, "From QoS to QoE: A tutorial on video quality assessment," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 2, pp. 1126–1165, Second Quarter 2015.

[30] S. Liew and D.-Y. Tse, "A control-theoretic approach to adapting VBR compressed video for transport over a CBR communications channel," *IEEE/ACM Trans. Netw.*, vol. 6, no. 1, pp. 42–55, Feb. 1998.

[31] N. Mohsenian, R. Rajagopalan, and C. A. Gonzales, "Single-pass constant- and variable-bit-rate MPEG-2 video compression," *IBM J. Res. Develop.*, vol. 43, no. 4, pp. 489–509, Jul. 1999.

[32] M. Carrascosa and B. Bellalta, "Cloud-gaming: Analysis of Google Stadia traffic," *Comput. Commun.*, vol. 188, pp. 99–116, Apr. 2022.

[33] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano, "A network analysis on cloud gaming: Stadia, GeForce Now and PSNow," *Network*, vol. 1, no. 3, pp. 247–260, Oct. 2021.

[34] P. Graff, X. Marchal, T. Cholez, S. Tuffin, B. Mathieu, and O. Festor, "An analysis of cloud gaming platforms behavior under different network constraints," in *Proc. 17th Int. Conf. Netw. Serv. Manag.*, 2021, pp. 551–557.

[35] O. S. Peñaherrera-Pulla, C. Baena, S. Fortes, E. Baena, and R. Barco, "Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks," *Sensors*, vol. 21, no. 4, Feb. 2021, Art. no. 1387.

[36] B. Hentschel, M. Wolter, and T. Kuhlen, "Virtual reality-based multi-view visualization of time-dependent simulation data," in *Proc. IEEE Virtual Reality Conf.*, 2009, pp. 253–254.

[37] F. Chiariotti, "A survey on 360-degree video: Coding, quality of experience and streaming," *Comput. Commun.*, vol. 177, pp. 133–155, Sep. 2021.

[38] Study on XR (extended reality) evaluations for NR, 3GPP, Technical Report (TR) 38.838, Dec. 2021. [Online]. Available: https://www.3gpp.org/dynareport/38838.htm

[39] W. Wu et al., "Dynamic RAN slicing for service-oriented vehicular networks via constrained learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2076–2089, Jul. 2021.

[40] S. Zhang, "An overview of network slicing for 5G," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 111–117, Jun. 2019.

[41] S. A. Kazmi, N. H. Tran, T. M. Ho, and C. S. Hong, "Hierarchical matching game for service selection and resource purchasing in wireless network virtualization," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 121–124, Jan. 2018.

[42] J. Zhang, W. Wu, and J. C. S. Lui, "On the theory of function placement and chaining for network function virtualization," in *Proc. 18th Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2018, pp. 91–100.

[43] R. L. Gomes, L. F. Bittencourt, and E. R. Madeira, "Reliability-aware network slicing in elastic demand scenarios," *IEEE Commun. Mag.*, vol. 58, no. 10, pp. 29–34, Oct. 2020.

[44] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.

[45] X. Shen et al., "AI-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 45–66, Jan. 2020.

[46] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement learning-based 5G network slice broker," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1543–1557, Aug. 2019.

[47] X. Chen et al., "Multi-tenant cross-slice resource orchestration: A deep reinforcement learning approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2377–2392, Oct. 2019.

[48] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, “Mobile traffic forecasting for maximizing 5G network slicing resource utilization,” in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[49] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “Resource sharing efficiency in network slicing,” *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 3, pp. 909–923, Sep. 2019.
[50] H. Jiang, T. Wang, and S. Wang, “Multi-scale hierarchical resource management for wireless network virtualization,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 4, pp. 919–928, Dec. 2018.
[51] L. Tang, X. He, X. Yang, Y. Wei, X. Wang, and Q. Chen, “ARMA-prediction-based online adaptive dynamic resource allocation in wireless virtualized network,” *IEEE Access*, vol. 7, pp. 130438–130450, 2019.
[52] C. Gutterman, E. Grinshpun, S. Sharma, and G. Zussman, “RAN resource usage prediction for a 5G slice broker,” in *Proc. 19th Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2019, pp. 231–240.
[53] N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, “Real-time network slicing with uncertain demand: A deep learning approach,” in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.
[54] A. Arfaoui, R. El-Azouzi, M. Haddad, and E. Sabir, “Flexible network slicing assisted 5G for video streaming with effective and efficient isolation,” in *Proc. 32nd Int. Teletraffic Congr.*, 2020, pp. 156–164.
[55] M. Chen, W. Saad, and C. Yin, “Virtual reality over wireless networks: Quality-of-service model and learning-based resource management,” *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5621–5635, Nov. 2018.
[56] M. Chen, W. Saad, C. Yin, and M. Debbah, “Data correlation-aware resource management in wireless virtual reality (VR): An echo state transfer learning approach,” *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4267–4280, Jun. 2019.
[57] X. Yang et al., “Communication-constrained mobile edge computing systems for wireless virtual reality: Scheduling and tradeoff,” *IEEE Access*, vol. 6, pp. 16665–16677, 2018.
[58] Video Codec SDK Documentation, Nvidia, Tech. Rep., Nov. 2022. Accessed: Mar. 15, 2023. [Online]. Available: https://docs.nvidia.com/video-technologies/video-codec-sdk/nvenc-video-encoder-api-prog-guide/
[59] S. M. Stigler, “Gauss and the invention of least squares,” *Ann. Statist.*, vol. 9, pp. 465–474, May 1981.
[60] R. Koenker and G. Bassett Jr, “Regression quantiles,” *Econometrica: J. Econometric Soc.*, vol. 46, pp. 33–50, Jan. 1978.
[61] C. Yu and W. Yao, “Robust linear regression: A review and comparison,” *Commun. Statist.–Simul. Comput.*, vol. 46, no. 8, pp. 6261–6282, Sep. 2017.
[62] R. M. Norton, “The double exponential distribution: Using calculus to find a maximum likelihood estimator,” *Amer. Statistician*, vol. 38, no. 2, pp. 135–136, May 1984.

Federico Chiariotti (Member, IEEE) received the PhD degree from the University of Padova, Padova, Italy, in 2019. He is currently an assistant professor with the Department of Information Engineering, University of Padova. Between 2020 and 2022, he worked as a post-doctoral researcher and as an assistant professor at the Department of Electronic Systems, Aalborg University, Aalborg, Denmark. He has authored more than 60 published papers, and his current research interests include network applications of machine learning, transport layer protocols, and virtual reality. He is currently an associate editor of the *IEEE Networking Letters*.

Matteo Drago (Member, IEEE) received the BSc, MSc, and PhD degrees in telecommunication engineering from the University of Padova, Padova, Italy, in 2016, 2019, and 2023, respectively. He visited Nokia Bell Labs, Dublin, Ireland, in 2018, working on QoS provisioning for 60 GHz networks. In 2021, he was a visiting scholar with the Department of Electrical and Computer Engineering, North Carolina State University (NCSU), Raleigh, North Carolina, working at the AERPAW testbed under the supervision of Robert W. Heath Jr. His research collaborations include companies (Huawei Munich Research Center) and research institutions (U.S. NIST, CTTC).

Paolo Testolina (Student Member, IEEE) received the BSc, MSc, and PhD degrees in telecommunication engineering from the University of Padova, Padova, Italy, in 2016, 2019, and 2023, respectively. He visited Nokia Bell Labs, Dublin, Ireland, in 2018, working on QoS provisioning for 60 GHz networks. In 2021, he was a visiting scholar with the Department of Electrical and Computer Engineering, North Carolina State University (NCSU), Raleigh, North Carolina, working with the AERPAW testbed under the supervision of Robert W. Heath Jr. His research collaborations include companies (Huawei Munich Research Center) and research institutions (U.S. NIST, CTTC).

Mattia Lecci (Member, IEEE) received the BSc degree in information engineering, the MSc degree in telecommunication engineering, and the PhD degree in information engineering from the University of Padova, Padova, Italy, in 2016, 2018, and 2022, respectively. During his academic journey, he gained valuable research experience as a guest researcher at the National Institute for Standards and Technology (NIST, 2018) and as a visiting scholar with Northeastern University (Boston, Massachusetts, 2021), and he currently serves as an R&D engineer with Keysight Technologies. His research focuses on channel modeling for mmWave frequencies, MAC scheduling for WiGig technologies, applied machine learning for communications, virtual reality traffic modeling, and improving QoE for multimedia signals.

Andrea Zanella (Senior Member, IEEE) received the laurea degree in computer engineering, and the PhD degree from the University of Padova, Padova, Italy, in 1998 and 2001, respectively. He is full professor with the Department of Information Engineering (DEI), University of Padova. He is one of the coordinators of the SIGnals and NETworking (SIGNET) research lab. His long-established research activities are in the fields of protocol design, optimization, and performance evaluation of wired and wireless networks.

Michele Zorzi (Fellow, IEEE) received the PhD degree in electrical engineering from the University of Padova, Padova, Italy, in 1994. After spending three years with the Center for Wireless Communications with UCSD, in 1998 he joined the School of Engineering of the University of Ferrara, Ferrara, Italy, where he became a professor, in 2000. Since November 2003, he has been with the Information Engineering Department, University of Padova. He was the editor-in-chief of several IEEE journals, and has served the IEEE Communications Society in various leadership roles.

Open Access provided by ‘Università degli Studi di Padova’ within the CRUI CARE Agreement