An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources

Mattia Lecci
leccimat@dei.unipd.it
Dept. of Information Engineering,
University of Padova
Padova, Italy

Andrea Zanella
zanella@dei.unipd.it
Dept. of Information Engineering,
University of Padova
Padova, Italy

Michele Zorzi
zorzi@dei.unipd.it
Dept. of Information Engineering,
University of Padova
Padova, Italy

ABSTRACT
Next-generation wireless communication technologies will allow users to obtain unprecedented performance, paving the way to new and immersive applications. A prominent application requiring high data rates and low communication delay is Virtual Reality (VR), whose presence will become increasingly stronger in the years to come. To the best of our knowledge, we propose the first traffic model for VR applications based on traffic traces acquired from a commercial VR streaming software, allowing the community to further study and improve the technology to manage this type of traffic. This work implements ns-3 applications able to generate and process large bursts of packets, enabling the possibility of analyzing APP-level end-to-end metrics, making the source code as well as the acquired VR traffic traces publicly available and open-source.

CCS CONCEPTS
• Networks → Network simulations.

KEYWORDS
Virtual Reality, Traffic Modeling, ns-3

ACM Reference Format:
Mattia Lecci, Andrea Zanella, and Michele Zorzi. 2021. An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources. In Proceedings of Workshop on ns-3 (WNS3 2021). ACM, New York, NY, USA, 7 pages. https://doi.org/xxx

1 INTRODUCTION
The growing demand for high performance telecommunication networks is driving both industry and academia to push the boundaries of the achievable performance.

The International Telecommunication Union (ITU) proposes requirements for International Mobile Telecommunication-2020 (IMT-2020) Enhanced Mobile BroadBand (eMBB) such as peak Downlink (DL) data rate of 20 Gbps with 4 ms user plane latency [7], for example by exploiting the large bandwidth available in the Millimeter Wave (mmW) spectrum. Similarly, Wireless Local Area Networks (WLANs) are also harvesting the potential of the mmW band with a family of standards known as Wireless Gigabit (WiGig), including IEEE 802.11ad and 802.11ay. While the former, first standardized in 2012 [2] and later revised in 2016 [11], is able to reach bit rates up to 8 Gbps, the latter is close to be officially standardized [13], and promises bit rates up to 100 Gbps.

These specifications for wireless systems enable a new generation of demanding applications such as high-definition wireless monitor, eXtended Reality (XR) headsets and other high-end wearable devices, center inter-rack connectivity, wireless backhauling, and office docking, among others [14].

In particular, XR, an umbrella name including technologies such as Virtual Reality (VR) and Augmented Reality (AR), has been targeted as a key application with growing interest in the consumer market [1]. Compact and portable devices with limited battery and computing power should be enabled to wirelessly support this type of demanding applications, to provide a fully immersive and realistic user experience.

To reach the limits of human vision, monitors with a resolution of 5073×5707 per eye with 120 FPS refresh rate will be needed [1]. These specifications suggest that rendering might be offloaded to a separate server as head-mounted displays should be light, silent, and comfortable enough to be worn for long periods of time. This poses significant strain on the wireless connection, requiring ~167 Gbps of uncompressed video stream. Clearly, real-time 360 video compression techniques allow to largely reduce the throughput requirements, at the cost of some processing delay, down to the order of 100–1000 Mbps.

While throughput requirements are already very demanding, low latencies are the key to the success or the failure of XR applications. In fact, many studies showed that users tend to experience what is called motion or cyber sickness when their actions do not correspond to rapid reactions in the virtual word, causing disorientation and dizziness [1, 5, 6, 8, 15]. Motion-to-photon latency is thus required to be at most 20 ms, translating into a network latency for video frames of 5–9 ms [1, 14].

In its simplest and most ideal form, raw XR traffic with a fixed frame rate \( F \) could be modeled by periodic traffic, with period \( 1/F \) and constant frame size \( S \) proportional to the display resolution. Real traffic, though, is first of all compressed with one of the many existing video codecs, and then properly optimized for real-time low-latency streaming, resulting in encoded video frames of variable size. Furthermore, the complexity of a given scene can also affect the time required to render it as well as the obtainable compression factor. Together, these factors make both the video frame...
Given the interest of both industry and consumers in XR applications and the peculiarity of the generated traffic, networks and networking protocols might be optimized to better support this type of traffic, ensuring its strict Quality of Service (QoS) requirements.

To the best of our knowledge, no prior state of the art on XR traffic modeling exists. Cloud gaming [3] was identified as a closely related problem, where a remote server renders and streams a video to a client with limited computational resources, which only feeds basic information to the rendering servers, such as keys pressed and mouse movements. The main difference with the problem under analysis is given by the more restrictive QoS constraints of XR applications, mainly due to the limits imposed by the motion sickness. Furthermore, in cloud gaming, client and server are often in different WLANs, making it harder to obtain reliable measurements on packet generation times. In fact, due to the specific constraints and requirements of XR applications, we expect the rendering server to be in a local network rather than being remotely accessed via the Internet.

Most works in the literature focus on network performance and limitations of cloud gaming [16], and we could find only two main contributions addressing traffic analysis and modeling. The authors of [4] provide a simple traffic analysis for three different games played on OnLive, a cloud gaming application that was shut down in 2015. The analysis focuses on packet-level statistics, such as packet size and inter-packet time and bit rate. They measured the performance of the streaming service under speed-limited networks, showing an evident frame rate variability. In [10], the authors tried to model the traffic generated by two games, also played on the OnLive platform. In particular, they recognized that video frames were split into multiple fragments, and re-aggregated them before studying their statistics. A number of DL and Uplink (UL) data flows were recognized, and characterized in terms of application packet data unit size and packet inter arrival time. Unfortunately, correlation among successive video frames was not modeled and the analysis referred to a single game played with an average data rate of about 5 Mbps.

The novelty of this paper can be summarized as follows:

1. To the best of our knowledge, this is the first generative model for APP-level XR traffic based on over 30 minutes of acquisition and processed VR traffic, with adaptable data rate;
2. we provide a flexible Network Simulator 3 (ns-3) module for simulating applications with bursty behavior and to characterize both fragment-level and burst-level performance;
3. we provide an implementation of the proposed XR traffic model, as well as a trace-based model, on the bursty application framework;
4. as a side contribution, several acquired VR traffic traces are made available, allowing researchers to (i) use real VR traffic in their simulations and (ii) further analyze and improve VR traffic models.

In the remainder of this work, we will describe the traffic acquisition and analysis in Sec. 2, propose a flexible XR traffic model in Sec. 3, discuss about the ns-3 implementation of the bursty application framework in Sec. 4, validate the model and show a possible use case in Sec. 5, and finally draw the conclusions of this work and propose future works in Sec. 6.

2 VR TRAFFIC: ACQUISITION AND ANALYSIS

The traffic model that will be described in Sec. 3 is based on a set of acquisitions of VR traffic. In the remainder of this section the acquisition setup and the statistical analysis of the acquired VR traffic traces will be presented.

2.1 Acquisition Setup

The setup comprises a laptop PC (equipped with an i7 processor, 16 GB of RAM, and an nVidia GTX 950M graphics card) acting as a rendering server, and transmitting the information to a smartphone acting as a passive VR headset. The two nodes are connected via tethering over USB 2.0 in order to avoid random interference from other surrounding devices and wireless channel fluctuations over Wi-Fi.

To stream the VR traffic, the rendering server runs the application RiftCat 2.0, connected to the VRidge app running on the smartphone.

Traffic traces were obtained using Wireshark, a popular open-source packet analyzer, running on the rendering server and sniffing the tethered USB connection. The traffic analysis was performed at 30 FPS, due to limitations of the rendering server, for target data rates of [5, 10, 15, 20, 25, 30, 35, 40, 45, 50] Mbps, and for a total of over 30 minutes of analyzed VR traffic.

2.2 Traffic Analysis

By analyzing the sniffed packet traces, we discovered that VRidge uses UDP sockets over IPv4, and that the UL stream contains several types of packets, such as synchronization, video frame reception information, and frequent small head-tracking information packets. In DL, instead, we found synchronization, acknowledgment, and video frame packets bursts.

Video traffic is, as expected, the main source of data transmission (Fig. 1a). Video frames are easily categorized by their transmission pattern: a single frame is fragmented into multiple smaller 127B packets sent back-to-back. By reverse engineering the bits composing the UDP payload, it was possible to identify 5 ranges of...
3 TRAFFIC MODEL

In this section, we will describe how we model frame sizes and inter-frame periodicity, leaving the model validation to Sec. 5.

3.1 Modeling Frame Periodicity

To fully characterize and thus generate a realistic frame period, more information is needed such as (i) the distribution of the frame period, (ii) the parameters of this distribution, (iii) the correlation between successive frame periods, and (iv) the correlation between the current frame size and the frame period.

To simplify the model, in this first analysis we assume the current frame size and the frame period to be independent, and we consider the stochastic process representing the frame period to be uncorrelated. We thus focus only on the distribution of the frame period.

For all the analyzed scenarios, the Cauchy distribution was found to best fit the data, closely followed by the Laplace distribution. Since the Cauchy distribution has no finite moments, i.e., no mean nor variance can be defined, we choose to model the frame periods as independent and identically Laplace-distributed random variables $X \sim \mathcal{L}(\mu, b)$ with Probability Distribution Function (PDF)

$$p(x|\mu, b) = \frac{1}{2b} \exp \left( -\frac{|x-\mu|}{b} \right),$$

where $\mu$ is the location parameter, $b > 0$ is the scale parameter, and $\mathbb{E}[X] = \mu$, $\text{std}(X) = b\sqrt{2}$.

Furthermore, Fig. 2a shows that the average standard deviation of the frame period over all acquired traces is about 5.39 ms, which is 16% of the mean frame period of 33.3 ms (for a frame rate of 30 FPS). This means that the periodicity of VR application, even for such an ideal setting, can be quite variable.

3.2 Modeling Frame Size

Following the discussion in Sec. 2.2, we propose to model the frame size distribution of the video frame with a Gaussian Mixture Model (GMM), i.e., $V(S) \sim \text{GMM}(\mu(S), \sigma^2(S))$, with PDF

$$V(S) = \chi_I(S)V_I(S) + (1 - \chi_I(S))V_P(S),$$

where $\chi_I(S)$ is the indicator function for I-frames which takes the value of 1 with probability $\omega_I(S)$ and 0 otherwise, and $V_I(S) \sim \mathcal{N}(\mu_I(S), \sigma^2_I(S))$, $f \in (I, P)$. Clearly, the fitted normal variable with the lower mean will be associated to P-frames while the one with the higher mean will be associated to I-frames.
To generalize the model, the parameters of the GMM should be extended to arbitrary target data rates.

Starting from the GMMs of the acquired traffic traces, means and standard deviations of I- and P-frames are generalized by fitting linear models as a function of the expected average frame size $S$. Since a target data rate approaching zero would require video frames to also approach zero, we force the linear fit to have no intercept, i.e., $\mu_I(S) = s_I S$, $\mu_P(S) = s_P S$, $\sigma_I(S) = d_I S$, and $\sigma_P(S) = d_P S$ as depicted with dashed lines in Figs. 2b and 2c.

By setting $\mathbb{E}[V(S)]$ equal to $S$ we get

$$
\begin{align*}
\mathbb{E}[V(S)] = w_I(S)\mu_I(S) + w_P(S)\mu_P(S) &= S, \\
 w_I + w_P &= 1,
\end{align*}
$$

from which $w_I(S) = w_I = \frac{1-w_P}{s_I-s_P}$ and $w_P(S) = w_P = \frac{s_I-1}{s_I-s_P}$, regardless of $S$. Setting $0 \leq w_I, w_P \leq 1$ results in a requirement for $s_I$ and $s_P$, specifically, $s_P \leq 1 \leq s_I$.

To make the model more robust, we first fit the GMM 50 times with random initial conditions and pick the best-fitting model, then weight the linear fit of the parameters proportionally to the goodness of the GMM fit.

It is important to note that during our analysis, we found that the traffic statistics are significantly different for low and high target data rates. It appears that the streaming application follows a heuristic trying to maximize the quality of experience, by changing the GoP and the compression ratio of I- and P-frames, making it hard to properly generalize the proposed model.

As shown in Figs. 2b and 2c, the following parameters have been extrapolated by the fitted models: $s_I = 1.243$, $s_P = 0.883$, $d_I = 0.139$, $d_P = 0.135$, $w_I = 0.326$, $w_P = 0.674$. The model is able to generalize sufficiently well the average frame size, especially for higher values of $R$, as expected, while the fitting of the standard deviations is much less accurate.

Frame sizes are independently drawn from the mixture model instead of simulating a GoP, since different GoPs were found for different target data rates, and always with a non-deterministic nature.

4 NS-3 IMPLEMENTATION

To properly model and test the performance of VR traffic over a simulated network, a flexible application framework has been implemented in ns-3 and made publicly available [9]. The framework is based on the ns-3.33 release, and aims at providing a novel additional traffic model, easily customizable by the final user.

The proposed framework allows the user to send packet bursts fragmented into multiple packets by a BurstyApplication, later re-aggregated at the receiver, if possible, by a BurstSink. Since the generation of packet bursts is crucial to model a wide range of possibilities, a generic BurstGenerator interface has been defined. Users can implement arbitrary generators by extending this interface, and three examples have been provided and will be described in Sec. 4.2. Finally, each fragment comprises a novel SeqTsSizeFragHeader, which includes information on both the fragment and the current burst, allowing the BurstSink to correctly re-aggregate or discard a burst, yielding information on received fragments, received bursts, and failed bursts.

More details on the implementation and the rationale behind these application will be given in the following sections.

4.1 Bursty Application

Inspired by the acquired traffic traces described in Sec. 2.1, the BurstyApplication periodically sends bursts of data divided into multiple smaller fragments of (at most) a given size. Since burst size and period statistics can be quite general, the generation of the bursts’ statistics is delegated to objects extending the BurstGenerator interface, later described in Sec. 4.2. A BurstyHelper is also implemented to simplify the generation and installation of BurstyApplications with given BurstGenerators to network nodes and examples are provided.

Each fragment carries a SeqTsSizeFragHeader, an extension of SeqTsSizeHeader which adds the information on the fragment sequence number and the total number of fragments composing the burst, on top of the (burst) sequence number and size as well as the transmission time-stamp. After setting a desired FragmentSize in bytes, the application will compute how many fragments will be generated to send the full burst to the target receiver, although the last two fragments may be smaller due to the size of the burst not being a multiple of the fragment size, and the presence of the extra header.

Traces notify the user when fragments and bursts are sent, while also keeping track of the number of bursts, fragments, and bytes
sent, making it easier to quickly compute some simple high-level metrics directly from the main of the simulation.

### 4.2 Burst Generator Interface

A generic bursty application can show extremely different behaviors. For example, an application could send a given amount of data periodically in a deterministic fashion, or the burst size or the period could be random with arbitrary statistics, successive bursts could be correlated (e.g., the concept of GoP for video-coding standards such as H.264 [12]), and even the burst size and the time before the next burst might be correlated.

To accommodate for the widest range of possibilities, a Burst-Generator interface has been defined. Classes extending this interface must define two pure virtual functions:

1. **HasNextBurst**: to ensure that the burst generator is able to generate a new burst size and the time before the next burst (also called next period in the remainder of this paper);
2. **GenerateBurst**: yielding the burst size of the current burst as well as the next period, if it exists.

Three classes extending this interface are proposed and briefly discussed in the remainder of this section, allowing users to generate very diverse statistics without the need to implement their own custom generator in most cases.

**Simple Burst Generator**. Inspired from OnOffApplication, SimpleBurstGenerator defines the current burst size and the next period as generic RandomVariableStreams. Users are thus able to model arbitrary burst size and next period distributions, by: using the distributions already implemented in ns-3; implementing more distributions; or simply defining arbitrary Cumulative Distribution Functions (CDFs) for EmpiricalRandomVariables.

Limitations for this generator lie in the correlation of the generated random variables: burst size and next period are independently drawn as are successive bursts.

**VR Burst Generator**. VrBurstGenerator is a direct implementation of the model proposed in Sec. 3, where bursts model video frames.

Similarly to the RiftCat software described in Sec. 2.1, this generator makes it possible to choose a target data rate.

While traces were taken at specific frame rates and target data rates, the proposed model attempts to generalize them, although without any knowledge on the quality of the generalization beyond the boundaries imposed by the streaming software.

To generate the frame size and the next period, LaplaceRandomVariable and MixtureRandomVariable have been implemented in ns-3.

A validation of the proposed model based on this burst generator will be discussed in Sec. 5.

**Trace File Burst Generator**. Finally, users might want to reproduce in ns-3 a traffic trace obtained by a real application, generated by a separate traffic generator, or even manually written by a user (e.g., for static debugging/testing purposes). For these reasons, TraceFileBurstGenerator was introduced, taking advantage of CsvReader to parse a csv-like file declaring a (burst size, next period) pair for each row. Once traces are imported, the generator will sequentially yield every burst, returning false as output to TraceFileBurstGenerator::HasNextBurst after the last row of the trace file is yielded, thus stopping the BurstyApplication.

A StartTime can be set as an attribute, allowing the user to control which part of the file trace will be used in the simulation. This can be especially useful when the total simulation duration is shorter than the traffic trace, making it possible to decouple users by setting different start times.

Several VR traffic traces using different frame rates and target data rates are available [9] in the described format, comprising some relevant meta data as part of the commented header. Interested readers can thus simulate real VR video traffic in their ns-3 simulations, or expand the analysis performed in Secs. 2.2 and 3.

### 4.3 Burst Sink

An adaptation of the existing PacketSink, called BurstSink, is proposed for the developed bursty framework. This new application expects to receive packets from users equipped with BurstyApplications and tries to re-aggregate fragments into packets.

While the current version of PacketSink is able to assemble byte streams with SeqTsSizeHeader, there are two reasons why BurstSink was created, specifically (i) to stress the dependency of this framework on UDP rather than TCP sockets, as the acquisitions suggested, thus expecting individual fragments sent unreliably rather than a reliable byte stream, and (ii) allow to trace the reception at both the fragment and the burst level.

The application implements a simple best-effort aggregation algorithm, assuming that (i) the burst transmission duration is much shorter than the next period, and (ii) all fragments are needed to re-aggregate a burst. Specifically, fragments of a given burst are collected, even if unordered, and, if all fragments are received, the burst is successfully received. If, instead, fragments of subsequent bursts are received before all fragments of the previous one are, then the previous burst is discarded. Information on the current fragment and burst can easily be recovered from the SeqTsSizeFragHeader, allowing the application to verify whether a burst has been fully received or not. If needed and suggested by real-world applications, future works might also introduce a concept of APP-level Forward Error Correction (FEC).

Traces notify the user when fragments are received and when bursts are successfully received or discarded, together with all the related relevant information. Furthermore, similarly to the BurstyApplication, also the BurstSink application keeps track of the number of bursts, fragments, and bytes received.

### 5 MODEL VALIDATION AND POSSIBLE USE CASES

This section will present a comparison between the acquired VR traces and the proposed model, as well as an example of use case.

For both the comparison and the example, we show the results of full-stack simulations highlighting the importance of accurately modeling a traffic source by using (i) the proposed model, (ii) the acquired traffic traces, and (iii) a simple deterministic model, which considers a burst size of $S = R/F$ and a period of $1/F$. We consider a simple Wi-Fi network based on IEEE 802.11ac, sending data over a single stream and using MCS 9 over a 160 MHz channel.
5.1 Model Validation

A comparison between the modeled distributions and the acquired traffic traces is shown in Fig. 3. In particular, the IFI shows a consistent distribution for almost all target rates, yielding good results with the simple Laplace distribution.

On the other hand, the frame size shows a less regular behavior among different target data rates, making it hard to fit well.

End-to-end results show good accordance between models and empirical data (see Fig. 4), except for the 95th percentile of the delay. In fact, the deterministic model is not able to fully catch the complexity of the traffic source, especially in the worst-case scenario, yielding overly optimistic results, while the trace-based simulation shows a clear behavior change starting from 30 Mbps, as already noted in Sec. 3.2. Instead, the proposed model shown in blue shows results closer to the traffic traces, mimicking an average behavior between low and high target data rates.

It is also important to notice the difference between fragment-wise and burst-wise statistics: focusing only on single fragments can lead to over-estimating the APP-level performance.

5.2 Examples of Use Cases

To exemplify the uses of the proposed model, we discuss a possible scenario of interest where we test how well an IEEE 802.11ac network can support VR traffic.

In Fig. 5, we show the simulation results of a scenario with multiple users running VR applications with a target rate of 50 Mbps in a Wi-Fi network. We compare the acquired trace file, where
Stations (STAs) import and generate disjoint parts of the trace file, with the proposed model and the simple deterministic model.

From Fig. 5a, it is possible to notice that the average burst delay is below the maximum tolerated delay of 9 ms [1] for up to 8 users. Instead, if a good overall quality of experience should be granted, the same bound for the 95\textsuperscript{th} percentile of the delay would only allow up to 4 users in the system, as shown in Fig. 5b, thus halving the network capacity for an increased reliability.

6 CONCLUSIONS

In this paper we presented a simple VR traffic model based on over 30 minutes of acquired traffic traces. While being simple, ignoring second-order statistics, and based on an ideal setting, this model marks a starting point for network analysis and optimization tailored for this novel and peculiar type of traffic. We also attempted to generalize the model to arbitrary target data rates.

Unfortunately, the analyzed VR streaming application showed an inconsistent behavior, making it hard to generalize it well and with high confidence. Nonetheless, simple full-stack simulations suggest that the model is still able to broadly capture the complex behavior of this type of traffic.

The proposed ns-3 framework for bursty applications is publicly available and open source [9], together with the implementation of the proposed traffic model and the actual traffic traces experimentally obtained.

Future works will focus on improving the quality and generality of this approach. For example, second order statistics will be taken into account, trying to better characterize the statistics of GoPs. More acquisitions will be taken, possibly longer, with different streaming and video encoding settings, on a number of VR applications, and on more powerful rendering servers thus reducing the impact of the server performance on the final model, while increasing diversity. Finally, more complex settings will be considered, e.g., adding head movements, in order to analyze possible correlations between them and the generated traffic.

ACKNOWLEDGMENTS

Mattia Lecci’s activities were supported by Fondazione CaRiPaRo under the grant “Dottorati di Ricerca 2018.” This work was partially supported by NIST under Award No. 60NANB19D122.

REFERENCES

[1] 2016. Empowering Consumer-Focused Immersive VR and AR Experiences with Mobile Broadband. White Paper. Huawei. https://www.huawei.com/en/industry-insights/outlook/mobile-broadband/insights-reports/vr-and-ar
[2] IEEE P802.11 Task Group ad. 2012. 802.11ad-2012 - IEEE Standard for Information technology – Telecommunications and information exchange between systems - Local and metropolitan area networks – Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band. (Dec. 2012). Superseded.
[3] W. Cai, R. Shea, C. Y. Huang, K. T. Chen, J. Liu, V. C. M. Leung, and C. H. Hsu. 2016. A Survey on Cloud Gaming: Future of Computer Games. IEEE Access 4 (2016), 7605–7620.
[4] M. Claypool, D. Finkel, A. Grant, and M. Solano. 2014. On the Performance of OnLive Thin Client Games. Multimedia Syst. 20, 5 (Oct. 2014), 471–484.
[5] E. L. Groen and J. E. Bos. 2008. Simulator Sickness Depends on Frequency of the Simulator Motion Mismatch: an Observation. Presence: Teleoperators and Virtual Environments 17, 6 (Dec. 2008), 584–593.
[6] L. J. Hettenger and G. E. Riccio. 1992. Visually Induced Motion Sickness in Virtual Environments. Presence: Teleoperators and Virtual Environments 1, 3 (Jan. 1992), 306–310.
[7] ITU. 2017. Minimum requirements related to technical performance for 5G radio interface(s). Report ITU-R M.2140-0. (Nov. 2017).
[8] H. G. Kim, W. J. Baddar, H. T. Lim, H. Jeong, and Y. M. Ro. 2017. Measurement of Exceptional Motion in VR Video Contents for VR Sickness Assessment Using Deep Convolutional Autoencoder. ACM Symposium on Virtual Reality Software and Technology (VRST), 7.
[9] M. Lecci. 2021. Implementation of the VR application model. (2021). Retrieved Feb. 2021 from https://github.com/signetlabdei/ns-3-vr-app.
[10] M. Manzano, M. Uretia, M. Sznitjević, E. Calle, J. A. Hernández, and M. Matija- sevic. 2014. Dissecting the Protocol and Network Traffic of the OnLive Cloud Gaming Platform. Multimedia Syst. 20, 5 (Oct. 2014), 451–470.
[11] IEEE P802.11i. 2016. IEEE Standard for Information technology – Telecommunications and information exchange between systems Local and metropolitan area networks – Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (Dec. 2016), 3534 pages.
[12] I. E. Richardson. 2010. The H.264 Advanced Video Compression Standard (2nd ed.). Wiley Publishing.
[13] Task Group ay. 2015. Status of Project IEEE 802.11ay. (Retrieved Feb. 2021 from http://www.ieee802.org/11/Reports/tgay_update.htm.
[14] Task Group ay. 2017. TGay Usage model. (Nov. 2017). https://mentor.ieee.org/802.11/dcn/15/11-15-0625-07-00ay-ieee-802-11-tgay-usage-scenes.pptx doc.
[15] S. von Mammen, A. Knote, and S. Edenhofe. 2016. Cyber Sick but Still Having Fun. In ACM Conference on Virtual Reality Software and Technology (VRST), 325–326.
[16] Z. Xue, D. Wu, J. He, X. Wei, and Y. Liu. 2015. Playing High-End Games in the Cloud: A Measurement Study. IEEE Transactions on Circuits and Systems for Video Technology 25, 12 (Dec. 2015), 2013–2025.