# Ultra-low Latency Point Cloud Streaming in 5G

Yannis Thomas and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics,
Athens University of Economics and Business, Athens 10434, Greece
{thomasi,xgeorge}@aueb.gr

***Abstract*—Point cloud streaming is a key component of 5G (and beyond) networks, serving as a foundation for holographic communication. While it enables immersive experiences, it also poses significant challenges to network infrastructure. Uncompressed point cloud streams produce Gbps traffic volumes, while compressed streams often suffer from multi-second latencies. Therefore, the feasibility of real-time applications that require ultra-low motion-to-eye latency, such as Network Music Performance and Remote Driving, remains unclear. In this work, we design and implement a novel point cloud streaming tool, based on the Draco encoder, Intel's RealSense SDK, and OpenGL. We deploy our prototype implementation in a private 5G stand alone network and provide an in-depth analysis of the latency and throughput of point cloud streaming. Our results show that deploying volumetric streaming services over 5G networks is still a challenging task. However, a combination of simple cost-reduction strategies can bring it significantly closer to feasibility.**

***Index Terms*—Ultra-low latency, Volumetric, Streaming, Draco.**

## I. Introduction

A *Point Cloud* (PC) in the context of volumetric video is a collection of 3D points that represent the shape and structure of objects or scenes in space [1]. Each point typically includes spatial coordinates (x, y, z) and may also carry additional attributes like color or reflectance. PC streaming is used to transmit dynamic volumetric content, such as people, environments, or objects, captured by one or more depth cameras, for real-time rendering in AR/VR/XR applications. Unlike stereoscopic 3D video, which offers a sense of depth from a single point of view, volumetric video can be rendered from different points of view.

PC streaming is gaining popularity as 5G and Beyond 5G networks become increasingly available [2]–[4]. These networks promise throughput and latency that can support previously infeasible services, such as holographic communication for human-to-human or human-to-robot interactions. Notable examples include Network Music Performance [5] and Remote Driving [6]; these require ultra-low latency communication, typically less than 100 ms of motion-to-eye latency, as well as high throughput, often up to hundreds of MBps.

While many studies have explored PC streaming performance, to the best of our knowledge, there are no detailed performance evaluations of volumetric streaming for ultra-low latency applications. Most published works exploit TCP [7]–[9], which offers recovery from packet losses, but due to its retransmissions, is unsuitable for ultra-low latency applications. Some works do employ UDP [10]–[12], but they convert the PC to a planar (2D) format, thus losing the ability to render it in 3D from different points of view. Of course, latency is not just due to network transmission: PC compression and decompression are computationally demanding tasks. Since the level of compression affects the bandwidth required for the PC stream, understanding the performance of PC streaming requires understanding the effects of the compression settings.

In this work, we design and implement a PC streaming tool and analyze in depth its performance in terms of latency, bitrate, and transmission resiliency. Our tool is built upon the Intel RealSense SDK for PC capture, the Draco encoder library for compression and decompression, a custom lightweight transport protocol, and OpenGL for rendering the 3D data. Most importantly, the tool allows individual analysis of the latency introduced by each stage in the PC processing pipeline, on both the producer and consumer sides. Our goal is to reveal performance bottlenecks and set directions for future research towards achieving ultra-low latency PC streaming.

We evaluate our tool in a private 5G *Stand Alone* (SA) network. We explore different strategies to adjust latency and bitrate, such as selecting the Draco compression level, lowering PC resolution, omitting color attributes and parallelizing compression. Our results offer a preliminary characterization of the conditions under which PC streaming becomes practical under real 5G conditions.

The remainder of the paper is organized as follows. In Sec. II, we discuss related work on PC streaming. In Sec. III, we introduce our novel streaming tool and present our evaluation setup, while in Sec. IV we present our evaluation results and discuss future research directions. Finally, in Sec. V, we summarize our conclusions.

## II. Background work

The literature on PC streaming is extensive, including several surveys. In [1], the authors deliver a comprehensive survey of volumetric streaming, providing a qualitative categorization of the approaches into Dynamic Meshes versus Time Varying Meshes, and Volumetric Mesh versus PC streaming. Emphasis is put on the computational cost of compression to reduce bitrate and its impact on latency, underscoring the benefits of exploiting *Mobile Edge Computing* (MEC).

Another survey covers latency and bitrate reduction methods [13]. The authors categorize designs into network-based and user-based. The network-based designs analyze network conditions, such as error-rate, to adapt the streaming setup. The user-based designs analyze user behavior, such as field

of view transitions, and proactively adapt the streaming setup. A survey specific to PC streaming is presented in [2]. The study covers work on encoding techniques, encompassing recent advancements in user-centric PC streaming, AI-driven PC streaming, and low-latency-focused PC streaming. It also points out some "transformative metaverse enablers", such as MEC, VLC, and network coding.

It is worth mentioning individually some papers that address PC encoding. In [14], the authors compare 4 codecs: Google's Draco, Corto, MPEG's Open 3D Graphics Compression (O3dgc) and OpenCTM, investigating the compression of vertex normals and attributes. The paper discusses live streaming, providing a simple model of the transmission pipeline to calculate an expected frame rate, as well as lower and upper bounds for end-to-end latency. The model suggests that Draco can deliver the lowest end-to-end latency (50-120 ms) for RTTs larger than 20 ms. In [15], the authors introduce an encoding scheme that reduces bitrate 55-99% with no quality loss, by transforming 3D PCs to 2D depth images, consisting of a color image and a depth map, and then compressing them using 2D encoders. However, they do not discuss the latency of this method. In [16], the authors deliver a QoE analysis of different encoding mechanisms, reporting that Draco requires higher data footprint to achieve the same mean opinion score as learning-based RS-DLPCC, while solutions that render PC to 2D video perform the best. Latency is also not discussed in this study.

Other papers focus on the design of protocols and services for streaming PCs in 5G networks. In [4], the authors discuss exploiting 5G edge computing in a function-as-a-service manner, where end-users upload the volumetric stream to a broker located in the 5G MEC, which transcodes media in a personalized and adaptive fashion. In [17] the authors investigate the addition of an MCU for the real-time delivery optimization of multiple PC streams. This design presents reduced computation and network requirements, at the cost of extra latency. ViVo [3] is a PC streaming platform for mobile end-users, covering the entire pipeline from capturing to rendering. It considers visibility limitations to determine which video content to fetch. It also considers parallel processing during decoding, which can increase the FPS on smartphone devices up to 400%.

The 3D streaming protocols tend to prioritize bandwidth economy, at the expense of latency, thereafter TCP is often used, despite its ill fit to real-time media transmission. In [8], the authors introduce a novel MPEG DASH-inspired technique for PC transmission which adapts streaming quality to connection quality. GROOT [7] is a streaming framework that enables real-time transmission and decoding on mobile devices. It introduces a novel tree data structure that enables parallelizing and, in turn, accelerating the PC decoding process. The authors report less than 100 ms motion-to-eye latency in a LAN. Yuzu [9] is similar to GROOT, using coding to reduce the bandwidth cost of 3D streaming by exploiting AI and perception-driven methods. Its evaluation shows that processing latency, without any transmission delays, can be less than 50 ms.

Other studies exploit UDP for transmission, to keep network latency below one RTT. To handle the large bitstream, those studies tend to convert PCs to a 2D format and incorporate reliability mechanisms, such as *Forward Error Correction* (FEC), to handle lost or late packets, especially in wireless networks. In [10], the authors introduce a network coding module for unequal error protection and FEC for 3D streams, which are encoded in 2D format. In [11], the authors introduce a FEC mechanism for stereoscopic 3D video, which consists of two 2D streams. In [12], the authors combine FEC with *Scalable Video Coding* (SVC) to transmit panoramic video, which is 3D spherical video converted into 2D. SVC is considered in [18], where the authors propose a multicast PC streaming protocol. The PC is encoded in different complementary layers that can be combined to achieve different levels of QoS, thus adapting to end-user needs and capabilities.

While UDP minimizes latency, it lacks the efficient control mechanisms for high-bandwidth transmission that TCP offers. Therefore researchers in [19] propose a best-of-both-worlds solution which exploits QUIC.

## III. Experiment setup

We developed and evaluated our PC streaming tool on a simple setup that consists of two applications, a producer and a consumer, running on different network nodes.[1] The producer captures, encodes, and transmits volumetric video to the consumer, which receives, decodes, and renders the stream. The nodes are connected in a peer-to-peer fashion using a 5G SA network; there is no *Selective Forwarding Unit* (SFU) between them, to keep communication latency as low as possible. In the following, we discuss individually the networking, streaming and software setup, and present the performance metrics explored in this study.

### A. Networking setup

The experiments took place at a private 5G testbed. The site offers an area of around 500 $m^2$ outdoor space which is covered by a private 5G SA network configured in the 3.7-3.8 GHz industrial spectrum. Band N78 has been exploited, a mid-band 5G band that is one of the most widely deployed for 5G due to its balance between coverage and capacity. Experimenting in a private 5G SA testbed means that the measurements are not affected by competing traffic, therefore presenting more accurate and clear results which, in turn, offer more trustworthy observations and conclusions.

The producer and consumer nodes were off-the-shelf Asus TUF A15 Ryzen 9 laptops running Ubuntu 24.04. The two nodes accessed the 5G network via the Gigabit Ethernet port of two Teltonika RUTX50 modems. DHCP was enabled to the modems, thus providing a private IPv4 address to the connected nodes. In the 5G network, the modems were using SIM cards that were assigned IPv4 addresses in a VLAN, that rendered the two modems visible to each other. To enable peer-to-peer connection at the producer and consumer nodes, port forwarding was enabled in the modems to push all traffic from the modem's VLAN address to the private node's address.

[1]Source Code is available at https://github.com/mmlab-aueb/nmp/tree/master/volumetric_streaming_app .

Thereafter, the two nodes were able to exchange packets using the VLAN addresses of their dedicated modems, without requiring NAT traversal.

TABLE I
CONNECTION CHARACTERISTICS IN PEER-TO-PEER MODE.

| Link | 5G band | Throughput (MBps) | One-way latency (ms) |
|---|---|---|---|
| 5G (SA) | N78 | 20.1 | 12.2 (st.dev. 1) |

The nodes had excellent signal quality, performing close to the theoretical optimal, as presented in Table I, which shows the characteristics of the peer-to-peer connection. The connection measurements were made via the `ping` and `iperf` Linux tools (using UDP), respectively.

It is worth noting that bandwidth measurements with `iperf` yielded quite different results for different transport protocols. Throughput using TCP was less than 8 MBps, while UDP delivered 20.1 MBps (or, more than 160 Mbps). After experimenting with our application, we discovered that excessively bursty transmissions led to significant losses at bitrates that were a fraction of the 5G link capacity; our video encoder was very bursty, as it emitted a very large number of packets after encoding each frame. We solved this problem by adding a 1 $\mu s$ latency between successive packets, thus shaping our traffic. We assume that the buffers of the RUTX50 modems were not large enough to handle the burstiness of our sender, or even common TCP connections that send packets in groups, especially in the slow-start phase.

### B. Streaming setup

The producer application (sender) was equipped with an Intel RealSense D435i depth camera connected via a USB type-C port. The consumer application (receiver) rendered the video in the laptop display, rather than AR glasses, as measuring latency when using glasses and headsets is very challenging. Further details on the streaming parameters are presented in the following subsection.

### C. Software setup

*1) Producer app:* The Producer application is implemented as a two-threaded C++ program, as shown in Fig. 1 (left side); one thread handles frame capturing, PC conversion and compression and the other thread handles PC transmission. During initialization, the capturing thread connects to the depth camera which is controlled via Intel's librealsense SDK,[2] and enters its main loop, which includes the following stages: capture a frame, encode it into a PC with the librealsense SDK, convert it to a Draco PC, compress it with the Draco library and push it to the other thread via shared memory. The sequential execution of the main loop stages means that the FPS rate is not fixed: it is determined by the processing latency at the main loop. However, this design facilitates measuring the latency of each individual stage.

During initialization, the transmission thread establishes a UDP socket for sending. Then, upon receiving a compressed PC from the capturing thread, it fragments it into 1400-byte chunks that fit in network packets and sends them to the receiver app. Transmission is achieved via UDP, exploiting a custom lightweight transport protocol to enable PC reconstruction. The protocol defines two header fields, a 4-byte frame sequence ID and a 4-byte chunk sequence ID, with a fixed 1400 byte payload. The receiver reconstructs the PC based on the chunk ID, which is multiplied by the chunk size to estimate the chunk's position within the frame. The receiver infers that a PC is "fully" transmitted when the first chunk for the next frame ID arrives.

Note that the transmission of PCs in narrowband links, like older Wi-Fi, can be significantly slower than the PC capture. The communication of the two threads is based on a circular buffer with a First-In-First-Out processing policy, which can lead to latency build-up due to extensive buffering. Therefore, our buffer size is limited to 10 PCs, dropping the oldest PC when the buffer is saturated. This restricts the buffering latency to 10 times the frame rate, which is roughly 0.33 s in case of 30 FPS.

*a) Multithreaded compression:* A PC can be broken into independent sub-PCs that can be (de)compressed individually, enabling parallelization and thus reducing overall compression latency. The producer implementation supports parallel compression of sub-PCs using multithreading, where each thread executes the "Draco pointcloud" and "compression" tasks of Fig. 1 for a different sub-PC. The use of multithreading is optional and the number of threads is configurable; when enabled, the Realsense PC is divided into $N$ equally sized sub-PCs, each assigned to one of the $N$ threads, which then generate $N$ compressed Draco PCs. These are subsequently reconstructed into a logically single PC and transmitted as one PC. The structure of the logically single PC consists of two fields per sub-PC: a 4-byte flag for sub-PC length and a byte-array for the sub-PC. This operation is transparent to all tasks but the compression and decompression, allowing seamless adaptation to different thread counts.[3]

*2) Consumer app:* The consumer is implemented as a two-threaded C++ program, as shown in Fig. 1 (right side); one thread handles reception and the other thread handles PC decoding and rendering. During initialization, the reception thread opens a UDP socket for chunk reception and enters its main loop, where it receives the chunks and reconstructs the compressed Draco PCs. The PCs are pushed to the decoding thread via shared memory. Reconstruction does not impose any buffering latency, as each PC is pushed when the first chunk of the next frame is received.

During initialization, the decoding thread creates a window for rendering the PCs and enters its main loop, where it waits for compressed PCs from the reception thread. When a PC is pushed, it is decoded using the Draco library and rendered using OpenGL. Currently, decoding and rendering (at the consumer) are conducted sequentially by the same thread, as these are faster than encoding (at the producer); this could change to improve performance, if needed.

[2]https://github.com/IntelRealSense/librealsense

[3]To enhance resilience to lost chunks, it is advised to list the sub-PC lengths in the first chunk of the PC, akin to an indexing table.

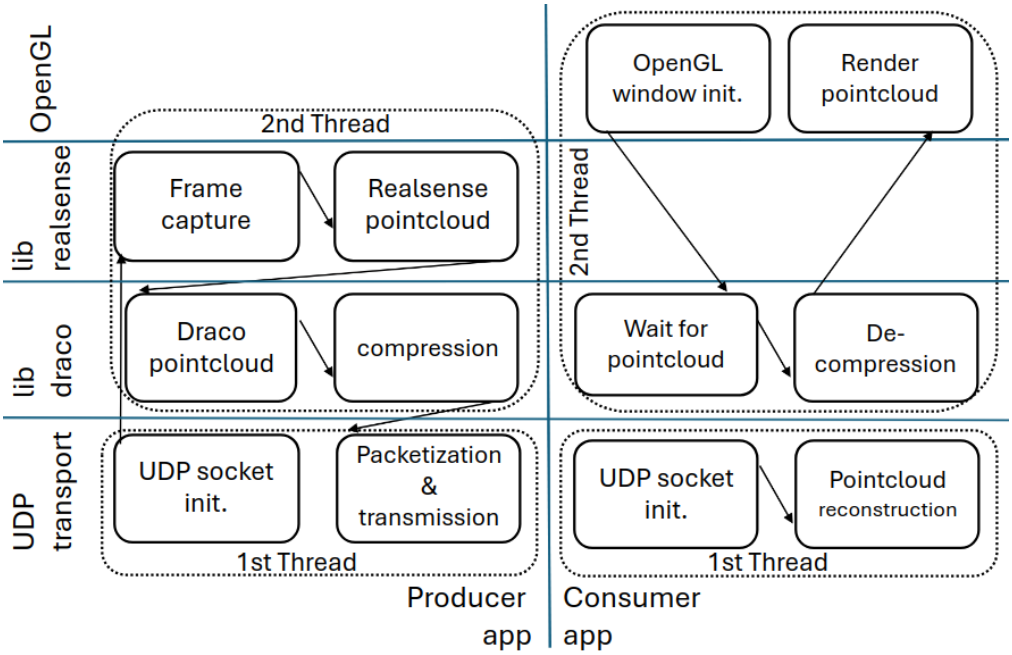


Fig. 1. Processing pipeline at the Producer (left) and Consumer (right) apps; each horizontal plane represents a technological layer.

Rendering is based on OpenGL using the GLFW library[4] in a 1280x720 window. For each point in the PC frame, the consumer draws the coordinates and the color using functions *glVertex3fv* and *glTexCoord2fv*, respectively. In addition, callback functions for mouse and keyboard events are implemented, allowing the user to customize the PC projection in real-time.

Currently, the consumer does not leverage multithreading for decompression. When the producer uses multithreaded compression, the consumer processes the individual sub-PCs sequentially. While a multithreaded consumer could reduce decompression latency, it would increase implementation complexity. Our preliminary evaluation suggests that the potential performance gain is limited in our setup; multithreaded decompression may be more beneficial in resource-constrained devices, such as smartphones.

### D. Scene and encoding setups

The PC frame in these experiments is captured at the default resolution of the Intel D435i (848×480), which is roughly 0.4 M points. Each point is represented by its coordinates (3×32-bit floats, quantized by Draco to 3×11-bit integers) and its RGBA color (4×8-bit integers). Compression uses the Kd-Tree Encoder with 16-bit quantization analysis of the Draco library.

The efficiency of Draco compression depends on the properties of the scene. Under the same compression level, the size of a compressed PC of a simple scene, e.g., a white wall 30 cm from the camera, is a fraction of the size of a complex scene, e.g., an overview of our 50 $m^2$ lab. Therefore, to provide a fair comparison of different encoding setups, all experiments were conducted using the same camera angle. The scene is close to what would be the case for music teaching, that is, a relatively complex scene with 1-2 m depth, which produced around 70% of the maximum PC size.

In the experiments, we test different encoding setups by controlling four parameters. First, we test the Draco compression levels, ranging from 1-10, with 1 being the fastest and 10 the most compact. Second, we modify the size of the PC stream, by dropping 25 or 50% of the points. Third, we experiment with different numbers of compression threads to reduce compression latency. Finally, we test performance with and without color information; the latter is denoted as a *colorless* PC.

### E. Performance metrics

We analyze performance via the following metrics:

- Processing Latency (ms): The total time to process a PC. At the producer, it shows the time from frame capture until the PC is put in the network. At the consumer, it shows the time from the receipt of the first chunk of the PC until the OpenGL window renders the PC.
- Compression Latency (ms): The time required to compress a frame using the Draco library.
- PC Size (Bytes): The size of a (compressed) frame.
- Transmission latency (ms): At the producer, it shows the total time to fragment a compressed PC into UDP chunks and put in in network. At the consumer, it shows the time for receiving all the chunks of a PC and reconstructing them into a PC; due to the consumer's design, this metric shows the inter-PC arrival time.
- End-to-end latency (ms): The time from the frame capture at the producer until the end of the rendering at the consumer.
- Reliability (%): The percentage of *entire* PCs delivered to the consumer without errors.

[4]https://www.glfw.org/

*a) Measurement method::* End-to-end latency and reliability metrics are estimated post-experiment by jointly analyzing the logs of the consumer and producer applications.[5] The clocks of the producer and consumer nodes are synchronized at the millisecond level using the NTP protocol over a secondary LAN connection specifically used for synchronization. The producer logs the time of capturing a frame and the consumer logs the time after a PC is rendered, thus allowing us to estimate the "end-to-end" latency. To assess transmission reliability, the producer and consumer log the hashes of the PC transmitted and received, respectively. Although partial PCs can be (partially) rendered, only flawless PC transmissions are considered successful.

The other measurements are straight-forward. The C++ applications log the system time between the execution of the different tasks to infer the individual latencies. Similarly, the PC size is determined by measuring the number of bytes pushed in the network.

*b) Frame rate estimation::* The FPS rate is not adjustable at the producer app; it depends on the processing latency, e.g., a processing latency of 33 ms leads to roughly 30 FPS. To allow bitrate comparisons, we define the bitrate for 30 FPS ($\text{bitrate}_{30}$) as the product of the compressed frame size (including transport overhead) with 30.

*c) Deployability limits::* We define latency and bitrate limits to assess the setup's deployability over real networks. In particular, the processing latency at both applications should be less than 33 ms (for 30 FPS), the bitrate should be at most 20 MBps (the uplink of our 5G SA links) and the end-to-end latency should be below 100 ms. Although these limits are not absolute, e.g., the FPS rate can be increased by parallel processing [3], they serve as a rough performance baseline.

## IV. Evaluation Results

### A. Draco compression

First, we explore the effect of Draco compression levels on system performance. Figure 2 includes three sub-figures that show the producer, consumer and shared metrics (from left to right), respectively, for 10 compression levels. The producer figure shows the PC frame size (right y-axis) and the transmission, processing and compression latencies (left y-axis) for different Draco compression levels in the producer app. The results verify that lower levels of compression deliver lower latencies and higher PC sizes, although the relation is not linear; the performance difference is most evident between levels 4 and 6. The results also reveal that the compression latency is roughly 80% of the overall processing latency, thus highlighting the importance of optimizing compression. Regarding compression efficiency, the difference between the fastest and slowest compression modes is 50% and 23% for compression latency and PC size, respectively. The transmission latency is affected by the PC size, since larger PCs require the emission of more UDP chunks. Due to the size of the processing latency (80-144 ms), frames are skipped, leading to a low FPS rate, which translates into a low bitrate (7.5-14.1 MBps), that can be transferred effectively from the 5G network. However, the (theoretical) $\text{bitrate}_{30}$ of these setups is 31-34 MBps, which is far beyond the capacity of the 5G link.

The consumer figure shows that the impact of the Draco compression level on the performance of the consumer application is minor. Processing latency, which comprises decompression and rendering latency, is almost indistinguishable for all levels. We do not show the rendering and decompression latencies separately, but they are roughly equal, each around 50% of the processing latency, or 25-30 ms. The transmission latency at the consumer, which shows the inter-PC arrival time, is identical to the processing latency at the producer, showing that the latter controls the rate of transmission.

Finally, the shared metrics show that reliability ranges between 96.8 and 98.3%, while end-to-end latency is 186-310 ms. The former verifies that the network can handle the low bitrate (7.5-14.1 MBps), required by our low effective FPS rate. The latter suggests that achieving low-latency volumetric streaming is feasible when sufficient network capacity is available, indicating that bandwidth remains the most critical challenge.

Overall, our measurements indicate that this basic setup is not capable of delivering good quality ultra-low latency volumetric streaming in the 5G testbed. The lowest $\text{bitrate}_{30}$ that Draco can offer is roughly twice the available capacity, the fastest processing latency at the producer is 2-3 times higher than what is required for 30 FPS, and the lowest end-to-end latency is twice the 100 ms deployability target.

### B. Removing color information

To assess the impact of color, we repeated the previous experiments, but discarding all color information before compressing the PC at the producer. Figure 3 includes three sub-figures that show the producer, consumer and shared metrics (from left to right), respectively, for compression levels 4, 6 and 8. The producer figure shows the color-less PC frame size (right y-axis) and the transmission, processing and compression latencies (left y-axis). Comparing these results against the results when color is included (Fig. 2), we can see that color accounts for roughly 50% of the processing latency and 66% of the resulting PC frame size and transmission latency, suggesting that color-less PCs are fairly inexpensive to compress and transfer. Regarding compression efficiency, when color is omitted, the difference between compression levels 4 and 6 is 28% and 3% for compression latency and PC frame size, respectively, suggesting that compression levels have a lower impact without color information. The $\text{bitrate}_{30}$ of the color-less modes is roughly 12.5 MBps.

The consumer and shared metrics offers a similar image, indicating that transferring color-less information is closer to being deployable. In color-less PCs, the consumer processing latency is 33 ms, supporting the 30 FPS goal, and the consumer transmission latency is equal to the producer processing latency, showing that the latter is the performance bottleneck. Finally, the 5G link is not saturated, offering 98.4% reliability, and 118-147 ms of end-to-end latency.

[5]The logs of the experiments presented in this paper are available at https://zenodo.org/records/15736910

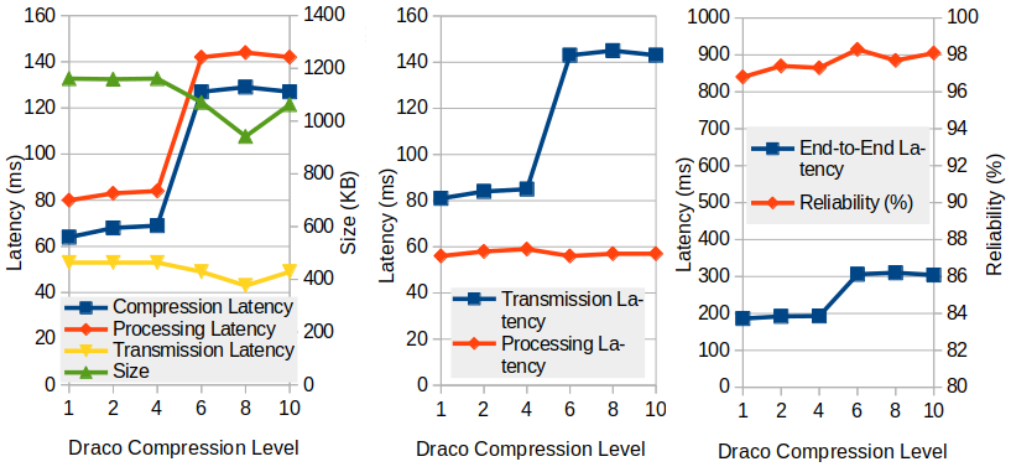


Fig. 2. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (1-10).

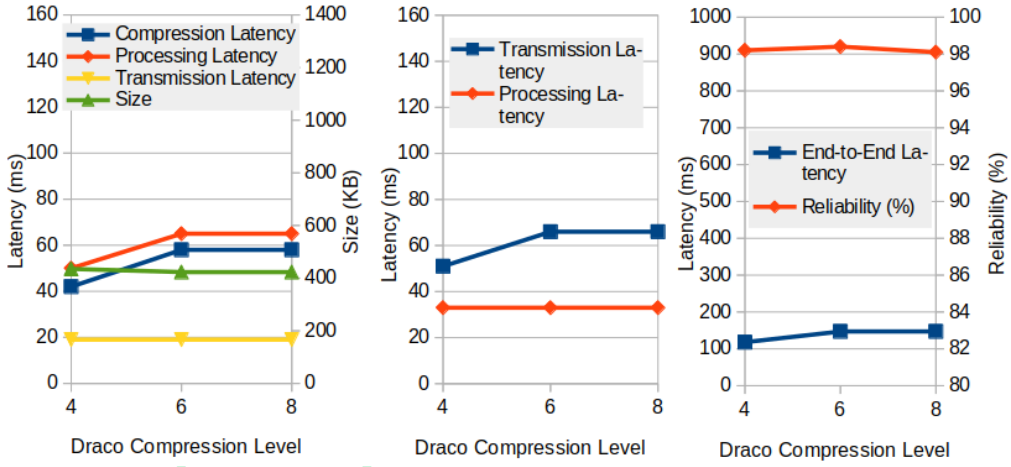


Fig. 3. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for color-less PCs.

Overall, operating in color-less mode significantly enhances the setup's deployability over real 5G networks, underscoring the importance of developing more efficient color compression methods for PCs.

### C. Reducing PC resolution

A different method to reduce latency and throughput requirements is to reduce the detail of the PC, therefore we decrease the PC resolution by dropping 25% and 50% of the points in each PC frame. The points are discarded before creating the uncompressed Draco PC at the producer; discarded points are uniformly distributed in the PC. We repeated the previous experiment with color information for compression levels 4-8, plotting the results in Fig. 4 and Fig. 5, respectively.

Compared to the basic setup (Fig. 2), the data indicate that reducing resolution leads to an almost linear decrease in compression latency and PC size. Therefore, if an X% reduction in the bitrate or latency is required, it can be accomplished via a roughly X% reduction in the resolution of the PC that is compressed and transmitted. The reduction is evident in the producer, consumer and shared metrics; reliability remains above 96% in all cases, showing that the induced bitrate fits within the 5G link. Additionally, the $\text{bitrate}_{30}$ is within the deployability bandwidth limit for all setups, except when compression level is 4 and resolution is 75% where the $\text{bitrate}_{30}$ is 23.2 MBps.

By reducing the resolution of the PC, the bandwidth requirement is satisfied by the 5G network capacity. However, the producer processing latency, which is higher than the 30 FPS limit, remains a significant performance bottleneck. End-to-end latency is close to the 100 ms limit, dropping to 112 ms in the best case. Overall, the 50% resolution reduction helps in rendering volumetric streaming realistic in 5G networks and may be considered a justified trade-off. Of course, the impact of this quality drop on QoE should also be investigated.

### D. Multithreaded encoding

To reduce the processing latency and increase the effective FPS, multi-threaded (de)compression is a promising mechanism. In this section, we experiment with different numbers of compression thread, namely, 2 and 3. For simplicity, we keep decompression single-threaded, since the consumer node is not

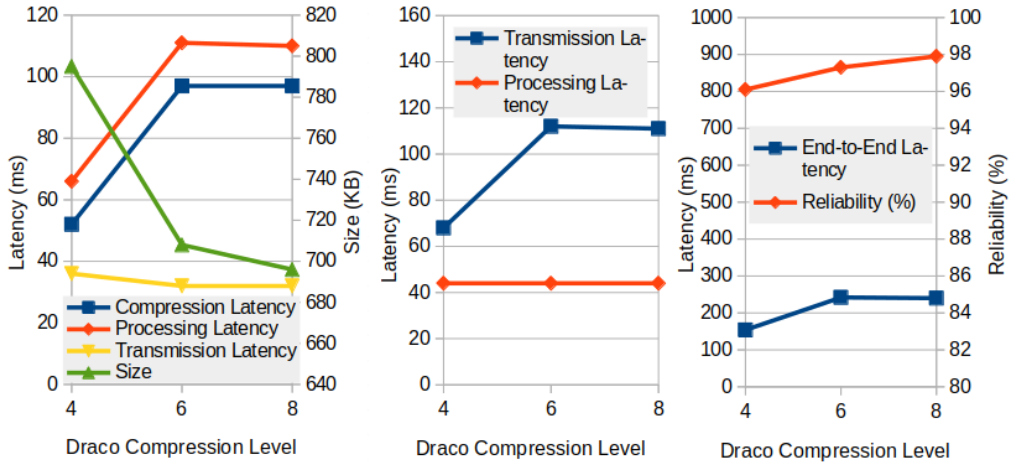


Fig. 4. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for 75% PC resolution.

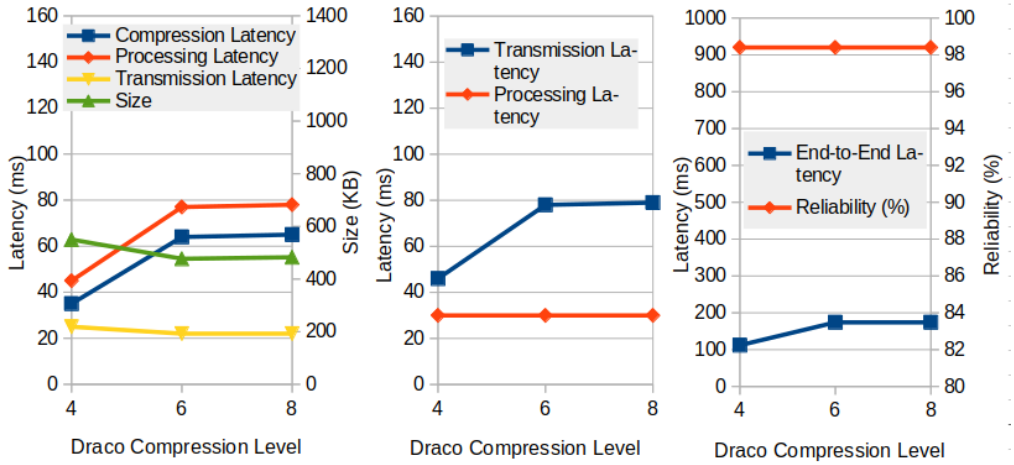


Fig. 5. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for 50% PC resolution.

the latency bottleneck in our setup. The results are depicted in Fig. 6 and Fig. 7 for 2 and 3 compression threads, respectively.

The most evident differences compared to the single-threaded setup are found in the producer metrics. Specifically, compression latency is linearly reduced, dropping by roughly 50% and 66% when 2 and 3 threads are exploited, respectively. This result verifies that multithreading can effectively reduce compression latency and, in turn, increase FPS. However, multithreaded compression penalizes compression efficiency, thus delivering larger PCs. In particular, the PC size is increased by up to 14% and 23% for 2 and 3 threads, respectively, resulting in a $\text{bitrate}_{30}$ of 31-38MBps. We hypothesize that the compression of multiple smaller sub-PCs is less effective than compressing a single large PC, since redundant or similar points that span across the PC could be isolated into separate sub-PCs. Altogether, by reducing the processing latency and increasing the PC size, the producer transmission latency has now become the performance bottleneck, requiring 49-60 ms.

Regarding the consumer metrics, the processing latency is not affected, since decompression and rendering remain single-threaded. We assume that equivalent reduction could be achieved through multithreading in the decompression latency, which accounts for roughly 50% of processing latency, that is 25 ms. On the other hand, the transmission latency (or inter-PC time) is significantly reduced, led by the new performance bottleneck, producer transmission latency.

Finally, the shared metrics provide interesting observations in both end-to-end latency and reliability. First, end-to-end latency is reduced by roughly 50% compared to single-threaded performance, verifying that reducing the producer's processing latency can have a significant impact on overall system performance. Second, in the case of compression level 4, the combination of low FPS and increased PC size produces a bitrate of 21.2 MBps that exceeds the available bandwidth. Consequently, severe packets loss is introduced, compromising reliability by roughly 10% and raising end-to-end latency in the order of seconds.

Overall, the use of multithreading in our setup is both a gift and a curse. It significantly reduces the producer processing latency, which is the performance and latency bottleneck, but it also increases the PC size, thus requiring more bandwidth resources than what the network can offer.

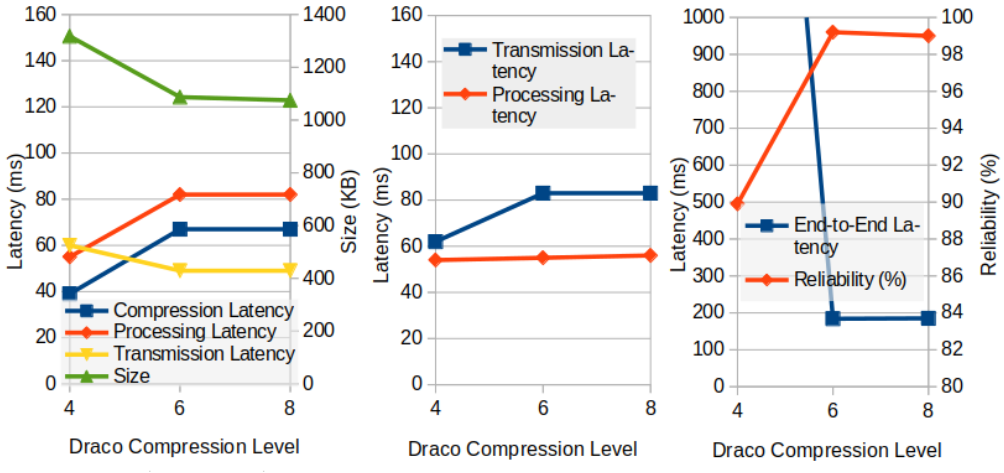


Fig. 6. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for 2 compression threads.

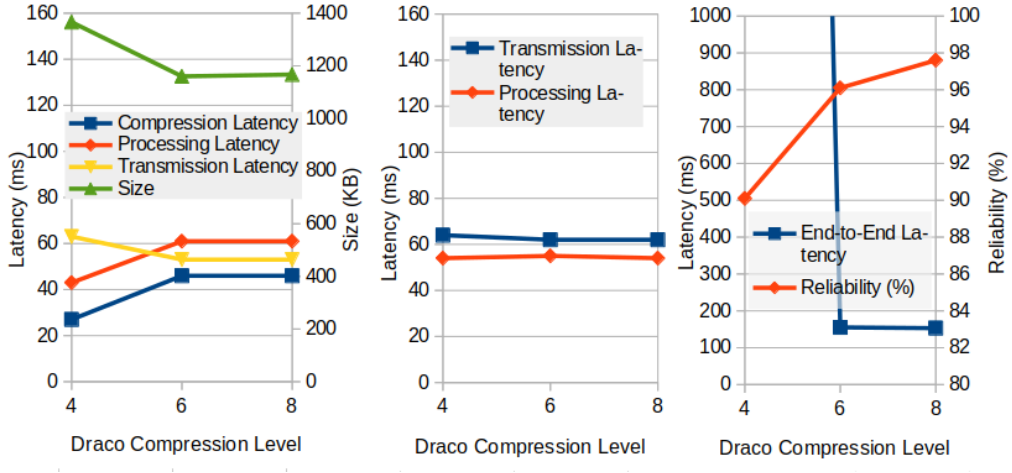


Fig. 7. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for 3 compression threads.

### *E. Combining methods*

Each of the previous methods present apparent trade-offs that limit their effectiveness. Combining different methods may offer a more balanced solution, hence we experiment with multithreaded encoding using 2 threads, and a 50% resolution reduction. The results are presented in Fig. 8.

The results suggest that performance can be greatly improved by combining multithreaded compression, to reduce processing latency, and resolution reduction, to control the traffic overhead caused by multithreaded compression. Specifically, the setup for compression level 4 is a promising solution towards ultra-low latency volumetric streaming in real 5G networks. The producer and consumer achieve a 30 FPS streaming rate and the 5G network supports the produced 18.7 MBps bitrate (and $\text{bitrate}_{30}$). Due to the deployable bitrate, reliability is 97.2% but end-to-end latency is 128 ms, near, yet above, the 100-ms deployability limit. It is worth noticing that, even though decompression is single-threaded, the multithreaded producer is still the performance bottleneck.

### *F. End-to-end delay variance*

Finally, we delve deeper in end-to-end delay in order to analyze and identify the key factors contributing to latency exceeding the deployability limit. Figure 9 depicts the end-to-end latency of the 1000 PCs for the setup that combines multithreaded compression and 50% resolution reduction (Fig. 8). The results suggest that the minimum possible end-to-end latency is roughly 80 ms. This coincides with the lowest individual processing and transmission latency measurements at the producer and consumer, which together sum to 80 ms as well. However, end-to-end latency exhibits significant variance in time (st.dev. is 40 ms), much larger than the variance of producer and consumer processing latency. Thereafter, it can be assumed that the network is not fast enough to steadily deliver low-latency packet tranmissions. The induced bitrate (18.7 MBps) approaches the maximum measured capacity of the 5G network, indicating that the network is operating near its highest theoretical performance.

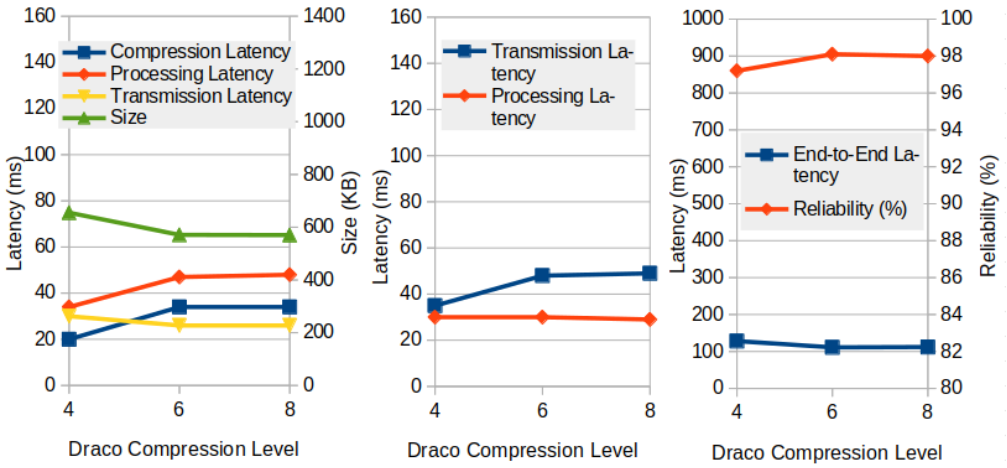


Fig. 8. Producer (left), consumer (center), and shared metrics (right) against Draco compression level (4-8) for 2 compression threads and 50% PC resolution.

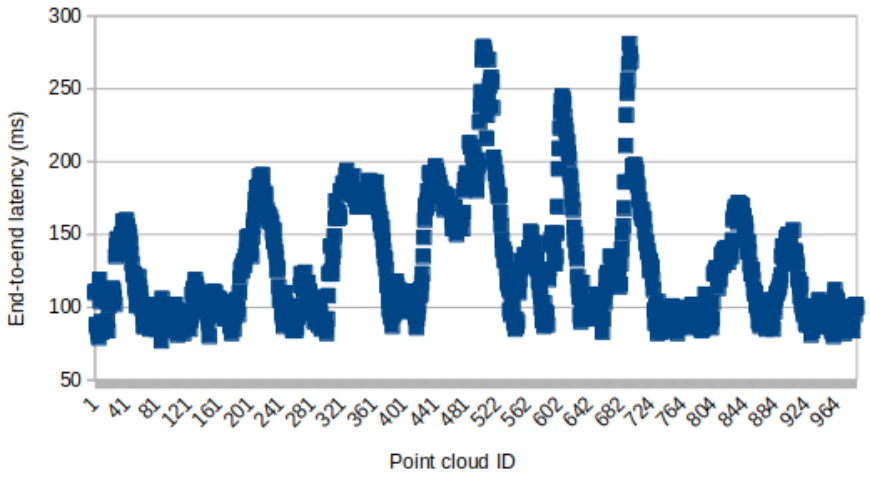


Fig. 9. Variance of end-to-end latency of PCs.

## V. Conclusions

We conducted a performance analysis of PC streaming for real-time applications that require ultra-low latency in a private 5G SA network. Even though we only scratch the surface of a multi-disciplinary topic which touches upon information theory, multimedia, computer graphics and networking, we provide some data points and future research directions for making volumetric streaming practical in 5G networks. In the following, we list the most notable outcomes of our work:

- Decompression performance is not significantly affected by the Draco coder's compression level.
- Assuming symmetric processing capabilities, the producer (encoder) is the performance bottleneck.
- Color is a significant part of the traffic footprint and the processing latency. Combining a color-less PC with inexpensive color compression, such as JPEG, could be a promising solution.
- Reducing PC resolution is effective in reducing communication cost. Techniques that identify and drop the most "insignificant" points within the PC are expected to have a positive impact.
- Multithreading can greatly reduce compression time, but at the cost of inflating the bitrate.
- Scene complexity has a significant impact on performance. If possible, simplifying the scene can a strong effect in reducing bandwidth requirements.
- A combination of the previous methods can make volumetric streaming achieve sub-100 ms latencies in 5G SA networks.
- Unregulated UDP and TCP can lead to underutilization of the 5G resources.

## Acknowledgment

The work reported in this paper has been partly funded by the EU through the subgrant Telepresence-Enhanced Network Music Performance (TENeMP, SPIRIT OC1) of project SPIRIT (grant agreement No. 101070672). The SPIRIT project has also received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI). The work has also been partly funded by the EU through the cascading action Adaptive Video Delivery for Network Music Performance (AViD-NMP, 6G-XR OC3) of project 6G-XR (grant agreement

No. 101096838). The 6G-XR project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU).

## REFERENCES


[1] I. Viola and P. Cesar, "Volumetric video streaming: Current approaches and implementations," *Immersive Video Technologies*, pp. 425–443, 2023.

[2] P. Enenche, D. H. Kim, and D. You, "On the road to the metaverse: Point cloud video streaming: Perspectives and enablers," *ICT Express*, 2024.

[3] B. Han, Y. Liu, and F. Qian, "Vivo: Visibility-aware mobile volumetric video streaming," in *26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–13.

[4] K. Konstantoudakis, D. Breitgand, A. Doumanoglou, N. Zioulis, A. Weit, K. Christaki, P. Drakoulis, E. Christakis, D. Zarpalas, and P. Daras, "Serverless streaming for emerging media: towards 5g network-driven cost optimization: A real-time adaptive streaming FaaS service for small-session-oriented immersive media," *Multimedia Tools and Applications*, vol. 81, pp. 12 211—12 250, 2022.

[5] L. Turchet, C. Rinaldi, C. Centofanti, L. Vignati, and C. Rottondi, "5g-enabled internet of musical things architectures for remote immersive musical practices," *IEEE Open Journal of the Communications Society*, 2024.

[6] C. Chen, Z. Lu, Y. Xu, S. Pang, and Y. Ma, "Research on real-time video transmission in intelligent vehicle outdoor remote driving system based on 5g network," in *International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 2024, pp. 16–21.

[7] K. Lee, J. Yi, Y. Lee, S. Choi, and Y. M. Kim, "Groot: A real-time streaming system of high-fidelity volumetric videos," in *26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[8] Z. Liu, Q. Li, X. Chen, C. Wu, S. Ishihara, J. Li, and Y. Ji, "Point cloud video streaming: Challenges and solutions," *IEEE Network*, vol. 35, no. 5, pp. 202–209, 2021.

[9] A. Zhang, C. Wang, B. Han, and F. Qian, "YuZu:neural-enhanced volumetric video streaming," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 137–154.

[10] H. Yang, X. Chen, X. Chen, B. Guo, S. Liu, X. Zhu, and Z. Li, "A 3D-DCT and convolutional FEC approach to agile video streaming," in *IEEE International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, 2022, pp. 636–644.

[11] D. You and D. H. Kim, "Combined inter-layer fec and hierarchical QAM for stereoscopic 3d video transmission," *Wireless Personal Communications*, vol. 110, no. 3, pp. 1619–1636, 2020.

[12] Y. Zhang, J. Zhang, Y. Huo, C. Xu, M. El-Hajjar, and L. Hanzo, "Scalable panoramic wireless video streaming relying on optimal-rate FEC-coded adaptive QAM," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 11 206–11 219, 2020.

[13] Y. Alkhalili, T. Meuser, and R. Steinmetz, "A survey of volumetric content streaming approaches," in *IEEE International Conference on Multimedia Big Data (BigMM)*, 2020, pp. 191–199.

[14] A. Doumanoglou, P. Drakoulis, N. Zioulis, D. Zarpalas, and P. Daras, "Benchmarking open-source static 3d mesh codecs for immersive media interactive live streaming," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 190–203, 2019.

[15] J. Zhong, H. Zhang, Q. Jia, J. Wu, P. Wang, H. Wang, L. Liu, X. Zhang, and Z. Guo, "Low-bitrate volumetric video streaming with depth image," in *SIGCOMM Workshop on Emerging Multimedia Systems*, 2024, pp. 39–44.

[16] J. Prazeres, M. Pereira, and A. Pinheiro, "Quality analysis of point cloud coding solutions," *Electronic Imaging*, vol. 34, pp. 1–6, 2022.

[17] G. Cernigliaro, M. Martos, M. Montagud, A. Ansari, and S. Fernandez, "PC-MCU: Point cloud multipoint control unit for multi-user holoconferencing systems," in *30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2020, pp. 47–53.

[18] A. Akhtar, J. Ma, R. Shafin, J. Bai, L. Li, Z. Li, and L. Liu, "Low latency scalable point cloud communication in VANETs using V2I communication," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[19] M. Palmer, T. Krüger, B. Chandrasekaran, and A. Feldmann, "The QUIC fix for optimal video streaming," in *Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2018, pp. 43–49.