# Low-Latency Volumetric Video Conferencing in Congested Networks Through L4S

Matthias De Fré[1], Jeroen van der Hooft[1], Chia-Yu Chang[2], Koen De Schepper[2], Patrice Rondao Alface[2], Danny De Vleeschauwer[2], Tim Wauters[1], Peter Steenkiste[3], Filip De Turck[1]
[1]IDLab, Ghent University - imec, Ghent, Belgium
[2]Nokia Bell Labs, Antwerp, Belgium
[3]Carnegie Mellon University, Pittsburg, United States of America

## ABSTRACT

Current networking solutions are unable to satisfy the low-latency requirements of real-time volumetric video conferencing when faced with heavy congestion scenarios. Traditional congestion controllers use packet loss or the change in round-trip time (RTT) to estimate the bandwidth. Commonly, this method is too slow as congestion has already occurred and the receiving user has already experienced a latency spike. Low latency, low loss and scalable throughput (L4S), recently published as RFC 9330, wants to alleviate this problem by aiming for sub 1 ms queuing delay for low-latency traffic by using accurate explicit congestion notification (AccECN) packet marking to notify applications of early congestion. We propose an L4S-based pipeline for volumetric video delivery, which achieves a more consistent latency under congestion compared to web real-time communication (WebRTC). In addition, L4S bandwidth estimation achieves a 45% faster convergence compared to Google congestion control (GCC) estimation, commonly used in WebRTC. Furthermore, in our detailed evaluation setup the L4S application experiences no packet loss, while the WebRTC-based version suffers from irrecoverable packet loss, resulting in 3% of frames being undecodable.

## CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → *Virtual reality*.

## KEYWORDS

Volumetric Video, Virtual Conferencing, Virtual Reality, WebRTC, L4S

**ACM Reference Format:**
Matthias De Fré[1], Jeroen van der Hooft[1], Chia-Yu Chang[2], Koen De Schepper[2], Patrice Rondao Alface[2], Danny De Vleeschauwer[2], Tim Wauters[1], Peter Steenkiste[3], Filip De Turck[1]. 2025. Low-Latency Volumetric Video Conferencing in Congested Networks Through L4S. In *ACM Multimedia Systems Conference 2025 (MMSys '25), March 31-April 4, 2025, Stellenbosch, South Africa.* ACM, Stellenbosch, South Africa, 11 pages. https://doi.org/10.1145/3712676.3714443


*MMSys '25, March 31-April 4, 2025, Stellenbosch, South Africa*

ACM ISBN 979-8-4007-1467-2/2025/03…$15.00
https://doi.org/10.1145/3712676.3714443

## 1 INTRODUCTION

The growing interest in virtual reality (VR) has led to its integration into various applications, such as virtual conferencing [26]. Traditional two dimensional (2D) implementations of these applications limited users to basic, non-immersive 2D perspectives. Proposals for six degrees-of-freedom (6DoF) video address this limitation by providing rotational and positional freedom [2]. However, current solutions fail to offer the low and consistent latency required for high interactivity, which is critical, as increased latency can lead to user frustration and a significant decline in the quality of the virtual conference experience [18].

The content of 6DoF applications can be divided into two distinct categories: image-based video and volumetric video formats [33]. Image-based solutions, such as light fields, generate a new frame based on numerous images to create a new view based on the user's viewing angle, captured by either a lenslet camera or a setup with many cameras [22]. However, this process leads to higher bitrates, increased latency and lower rendering performance at the client-side [33]. In contrast, volumetric video uses three dimensional (3D) objects such as meshes, consisting of vertices and a texture, or point clouds, consisting of individual points with a color, to represent objects within the video [4]. Capturing these objects can be done using a single specialized camera, resulting in a single red, green, blue and depth (RGBD) image, or with multiple specialized camera views that must be stitched together to create a single 3D object [3]. Compared to image-based video, volumetric video-based approaches yield bitrates that are significantly lower. Furthermore, the use of existing advanced rendering techniques increases overall quality. The simplicity of the capture process, together with its superior bandwidth and lower computational complexity, makes volumetric video an ideal candidate for 6DoF videos.

To ensure low latency in streaming, various protocol suites like low-latency dynamic adaptive streaming over HTTP (LL-DASH) and low-latency HTTP live streaming (LL-HLS) are often considered, yielding latencies ranging from 1 to 5 seconds in live streaming scenarios [33]. For true real-time communication, user datagram protocol (UDP)-based solutions like web real-time communication (WebRTC) are preferred, aiming to minimize latency by eliminating chunking overhead caused by LL-DASH and LL-HLS segments [32]. Additionally, WebRTC-based applications enable transmission of encoded frames through push-based methods, rather than the pull-based approach of transmission control protocol (TCP) formats, further improving the overall latency [5, 25].

The chosen streaming format alone does not guarantee both low latency and high quality of experience (QoE). Due to varied network conditions among users, content quality has to be adapted

to available bandwidth to ensure real-time delivery without packet loss. Unlike traditional video, volumetric video offers numerous viewing angles, allowing for quality adjustments based on the user's field of view (FoV) and position. Higher quality can be allocated to objects in proximity or directly within the FoV, while peripheral or distant objects are delivered at a lower quality [34, 14].

Quality adaptation relies on accurate bandwidth estimation. Traditional TCP methods typically employ a capacity-seeking approach: the bandwidth is progressively increased until packet loss is encountered, after which the transmission window is reduced [7]. Alternatively, other approaches exists which leverage variations in round-trip time (RTT) as indicators of network congestion [9]. For UDP, estimation happens at the application level using loss-based feedback messages that are exchanged between the sender and receiver. However, classic estimators react after congestion has taken place. In contrast, low latency, low loss and scalable throughput (L4S) aims to keep queuing delays below 1 ms by marking packets with accurate explicit congestion notification (AccECN) whenever there are early signs of congestion. This mechanism ensures prompt congestion response and consistent low latency [7]. Although the behavior of L4S is advantageous for real-time applications, it has yet to be evaluated for volumetric video conferencing.

The contributions of this paper are twofold. First, we propose an L4S-based streaming pipeline to support real-time communication for immersive applications, with a focus on maintaining a low latency in congested networks. This pipeline delivers consistent and uninterrupted volumetric video conferencing without high-latency spikes, which is essential for enhancing the user experience.. Second, we perform a comparative analysis between our proposed pipeline and an open-source WebRTC-based volumetric video solution [14], evaluating their respective performance in terms of latency, quality and behavior of the bandwidth estimator. The results show that the L4S-based solution achieves a similar, but more consistent, latency, faster bandwidth estimator convergence, and no packet loss, compared to the WebRTC-based approach when faced with competing traffic.

The remainder of this paper is organized as follows. Section 2 provides an overview of the current state of the art concerning volumetric video streaming. Section 3 introduces our proposed L4S-based approach to facilitate low-latency streaming of volumetric video. Section 4 presents the evaluation methodology and discusses the obtained results. Finally, Section 5 concludes the paper.

## 2 BACKGROUND AND RELATED WORK

This section presents an overview of the related work on real-time streaming of volumetric video. First, we introduce the concept of volumetric video and how it differs from traditional 2D video. Following this, we describe traditional transport protocols and solutions and the problems that occur with these solutions. Then, we introduce L4S-based transport as a possible solution to existing transport problems. Finally, we give an overview of the currently used quality adaptation methods for volumetric video streaming.

### 2.1 Volumetric Video

Because of the added depth dimension, volumetric video presents a substantially higher level of complexity in both transmission and encoding compared to traditional 2D video. The added complexity presents a challenge in maintaining low latency, making it essential to carefully select the type of volumetric content and encoder to ensure smooth, real-time video delivery.

While most 3D applications typically utilize meshes due to their advanced feature set, significant challenges emerge when applying them to volumetric video. The primary issue is that mesh capturing and processing introduce excessive latency, rendering them unsuitable for real-time virtual conferencing. In contrast, point cloud capture offers a more efficient alternative, as it eliminates the need to convert the captured data into a mesh. Additionally, it avoids computationally expensive mesh optimizations aimed at reducing object complexity [1]. Furthermore, point clouds have proven effective in volumetric video delivery. VR2Gather, a real-time volumetric video application, uses point clouds to enable real-time, bidirectional experiences, such as immersive museum visits or remote consultations [35].

No matter the representation, volumetric video typically results in bitrates that remain impractical for contemporary networks. For instance, large point clouds, such as those in the 8i dataset [12], can reach up to 5.2 Gb/s [33]. To address this problem, numerous point cloud codecs have been proposed to compress the data [20]. However, in virtual conferencing, minimizing encoding latency is essential to achieving a frame rate above 15 FPS, which is necessary for an acceptable user experience. As a result, highly bandwidth-efficient codecs such as video point cloud coding (V-PCC) and geometry-based point cloud compression (G-PCC) are unsuitable, as they have encoding and decoding times of multiple seconds for a single frame [38]. Alternative codecs, including cwipc [11] and Draco [16], achieve single-frame encoding times below 100 ms, which is viable for real-time volumetric video conferencing, albeit at the cost of increased bandwidth consumption [14]. However, these resulting bitrates are low enough to support volumetric video conferencing with a limited number of users. Furthermore, as the encoded frames are still quite large, they will need to be fragmented into a large number of packets. Since neither codec supports partial decoding, every packet must be received for successful playback. This requirement becomes challenging if many packets are dropped due to network congestion, highlighting the importance of selecting an appropriate network protocol.

### 2.2 TCP-based vs UDP-based content delivery

For on-demand video, HTTP-based implementations are often preferred due to their inherent reliability, ensuring that the whole video can be received by the user. Within the realm of HTTP streaming, protocol suites such as dynamic adaptive streaming over HTTP (DASH) and HTTP live streaming (HLS) have gained widespread adoption. These formats segment video content temporally and encode these segments at various bitrates and quality levels. This approach introduces a latency ranging anywhere from 1 to 18 seconds for live streaming scenarios [37], rendering it unsuitable for real-time applications. In contrast, WebRTC, a real-time communication suite utilizing UDP, achieves sub-second delays, making it a more suitable choice for real-time volumetric video streaming [25].

Unlike TCP-based solutions, such as LL-DASH, UDP-based approaches lack inherent flow control and bandwidth estimation

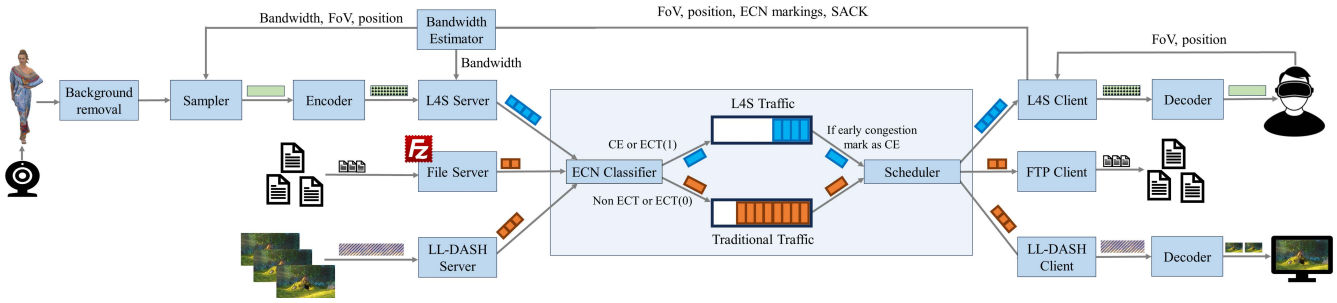



**Figure 1: System architecture for an L4S-based volumetric video streaming solution sharing the network with non-L4S flows.**

mechanisms. As a result, congestion controllers such as Google congestion control (GCC) [10] and Self-Clocked Rate Adaptation for Multimedia (SCReAM) [24], are implemented at the application layer to fulfill the purpose of bandwidth estimation and flow control. The primary objective of these controllers is to maintain a good balance between latency and high throughput by accurately estimating available bandwidth and pacing packets accordingly [13]. Similar to DASH-based solutions, the application can then use the bandwidth to select the transmitted quality on a per-peer basis, so that the video stream remains below the supplied bitrate [14]. In scenarios with a limited number of users, a unique quality representation can be generated for each user based on their available bandwidth, with the video encoder adjusting compression or reducing quality as needed to keep the stream within the bandwidth constraints of each user [27]. However, this approach significantly limits scalability, as the need to create a distinct representation for each user constrains the maximum number of participants. In this paper, the primary focus will be on the individual encoder to better illustrate the bandwidth estimation. Furthermore, for volumetric video, quality adaptation can be further enhanced by using the position and rotational information to prioritize the most important details [34].

For traditional transport, a significant portion of the latency comes from the queuing delay, with increased throughput raising the likelihood of higher, more variable queuing delay [29]. This is caused by congestion and queue buildup, particularly when all flows attempt to use their maximum bandwidth. Congestion eventually results in queue overflow and packet loss until the queue begins to drain [17]. The packet loss and increased delay are signals to the congestion controller that it should reduce its send rate in order to reduce network congestion. However, flows quickly increase send rates, restarting the process. Additionally, the queuing discipline at bottleneck links affects congestion reaction time; classic active queue management (AQM) disciplines delay packet drops for up to 20 ms after a burst before dropping the packet, potentially exacerbating delay [7]. These slow adaptation times, compounded by loss, cause buffering, increased latency and frame loss in streaming until network flows finally adjust their send rates. In the context of volumetric video, this increased buffering and latency can contribute to cybersickness [31]. Furthermore, due to the high bitrates of volumetric video, packet loss has a greater impact. Even a small percentage of packet loss can necessitate retransmitting a significant number of packets, each with its own risk of being lost again, resulting in further delays.

## 2.3 L4S-Based Transport

The emergence of real-time interactive applications like cloud gaming and virtual reality has intensified the demand for network protocols that prioritize both low latency and high throughput [8]. Traditional streaming solutions and congestion control mechanisms often struggle to achieve this balance, as latency reduction can come at the expense of reduced throughput, leading to a reduced user experience [19].

L4S attempts to address the problems that occur in highly congested networks. At the protocol level, L4S uses either TCP or UDP to transport data and acts as a congestion controller for these protocols. L4S also operates within the network architecture, enabling the identification and prioritization of latency-sensitive traffic flows by using the AccECN field of network packets [30]. This approach allows for differentiated services, ensuring minimal queuing delays for these flows without compromising overall network efficiency too much. L4S leverages AccECN to signal early congestion to senders, enabling them to adjust their transmission rates dynamically before the queuing delay becomes too high. This early congestion notification allows senders to react faster to upcoming congestion events, and prevents the queue from overflowing which would result in packets being dropped. Having a low queuing delay target also ensures that the latency within the network is less variable compared to traditional queuing systems [7].

To ensure coexistence with existing traffic, a dual queue setup is commonly used [15]. With this type of AQM, it becomes possible to split low-latency traffic from classic network traffic using the AccECN field. The queue containing the classic traffic works as a traditional queue, filling up until eventual packet loss. The low-latency queue will not immediately drop traffic, but will instead mark packets as congestion experienced (CE). Additionally, to prevent bursty traffic in the queue, each L4S application is required to space it's packets equally in a pacing interval. In the context of volumetric video, which has large bitrates, this pacing interval should be chosen carefully to maintain a balance between low latency and burstiness. In this paper, we will leverage L4S-based transport to create a volumetric video pipeline. To the best of the authors' knowledge, this is the first paper to study and evaluate L4S-based volumetric video in detail.

# 3 L4S-BASED VOLUMETRIC VIDEO DELIVERY

In this section, we first present the proposed volumetric video delivery architecture, followed by an explanation of the employed

quality adaptation scheme. Next, we provide an overview of the L4S components used, along with their implementation details.

## 3.1 Volumetric Video Delivery Architecture

The first step of the pipeline shown in Figure 1 involves capturing of the volumetric video, typically a person in virtual conferencing scenarios. In the context of this paper, it is assumed that a single camera is utilized, although this framework can be easily extended to accommodate multiple cameras by incorporating an additional component that is responsible for merging the views from all cameras. A specialized depth camera captures an RGBD image of the user, which is subsequently converted into a point cloud representation. This process may include an optional sampling step to reduce the number of points in the point cloud. However, caution must be exercised to ensure that sampling does not introduce a significant amount of latency. After sampling, it is necessary to encode the point cloud to reduce the bitrate, ensuring efficient transmission in networks with lower bandwidths. Similar to the sampling component, the encoding process must be performed with minimal latency to allow for higher frame rates, provided that sufficient bandwidth is available. Furthermore, it is important to note that current real-time state-of-the-art point cloud codecs do not support partial encoding. As a result, packet loss has a more pronounced effect, as the entire frame must be received for successful decoding.

After encoding, the frame is processed through the L4S network stack, beginning with the server component. Each frame is divided into packets, each containing frame data and a header for reconstruction at the receiver. The server is tasked with ensuring compliance with L4S requirements, such as implementing pacing and L4S-compliant bandwidth estimation. The next stage of the L4S stack is the queuing mechanism of the network, which must differentiate between L4S and traditional network traffic, handling each traffic type accordingly. This process may involve assigning different priorities or allocating distinct bandwidth shares to each flow type. For L4S traffic, it is critical to implement mechanisms that marks L4S packets so that they can anticipate congestion early enough to mitigate its effects. In contrast, traditional flows, such as large file downloads or LL-DASH streaming, typically have less stringent latency requirements than real-time virtual conferencing, which demands minimal latency. Further details regarding the specific implementation of the L4S components are discussed in the following subsections.

Upon delivery, the frames are decoded and displayed to the user. In most cases, rendering is carried out using a game engine, such as Unity, on a high-performance desktop PC. The rendered content is then transmitted to a head-mounted display (HMD) either through a high-speed wired connection or via a wireless solution, such as 5 GHz Wi-Fi or 5G.

## 3.2 Bandwidth Estimation

Bandwidth estimation is performed by leveraging the number of marked packets to calculate the marking probability. Based on this information, and using a target RTT of 25 ms, the system adjusts the bandwidth, either increasing or decreasing it accordingly. In the event of packet loss, either due to severe congestion or suboptimal network conditions, the L4S congestion controller falls back to a classic TCP-Reno congestion controller. As a result, the estimated bandwidth is halved upon detecting packet loss, helping to maintain low latency by reducing the need for retransmissions beyond the already lost packets. However, this approach leads to a reduction in video quality due to the lowered throughput.

In comparison to GCC, this L4S-based approach is more sensitive to network instability. GCC employs a lower threshold for packet loss, which mitigates the impact of minor losses that result from faulty equipment [36]. This lower boundary allows GCC to maintain stability in the presence of occasional packet loss, whereas the proposed approach responds more aggressively in such conditions.

## 3.3 Packet Acknowledgment

The delivery of packets is confirmed using selective acknowledgments (SACK) messages. In addition to containing the acknowledgments, it also includes information on the number of bytes marked as CE. Subsequently, these AccECN markings, along with the target RTT, are utilized by the congestion controller to estimate the available bandwidth. Moreover, the data contained within the SACK message enables the application to identify which packets require retransmission in the event of packet loss.

## 3.4 Quality Adaptation

An essential aspect of the architecture, present across multiple components, is the concept of quality adaptation. The objective is to optimally calculate the highest achievable quality within the constraints of limited available bandwidth. In the case of volumetric video, this calculation must also account for the user's FoV position and rotation. Achieving this, however, necessitates the coordination of several components that exchange information with one another.

In our solution, point cloud frames are adapted using an existing individual encoding approach [14]. This encoding method determines the required sampling rate for each user and uniformly reduces the number of points in the point cloud according to this rate. This approach facilitates a smooth scaling of frame size and enables more effective comparisons between the employed congestion control algorithms. Furthermore, it produces streams that are visually comparable to those generated by a multiple description coding (MDC)-based approach for use cases with up to 10 users, at a lower bitrate [14]. Alongside the estimated bitrate, both the user's position and FoV are required. This data is transmitted from the client to the server along with packet acknowledgments within the SACK messages. The positional and rotational information is utilized by the sampler to limit the maximum achievable quality produced by the individual encoding approach. The sampler achieves this by dividing the user's FoV into multiple equal-sized regions, each with a defined maximum sampling rate, and then determining the region in which the object currently resides. This approach is based on the rationale that the objective quality of the object can be reduced without significantly affecting the perceived visual quality for the user. However, it is important to note that there will be a discrepancy between the used position and the current position of the user as the pipeline does not perform any viewport prediction. Additionally, the transmission of the quality decision, and the time for the next frame to become available (between 1 ms and 33 ms for 30 FPS) both add latency before the new quality becomes visible.

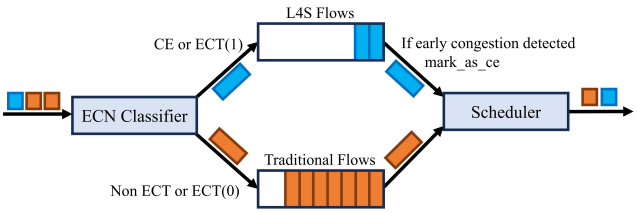



**Figure 2: L4S compliant queue separating L4S and traditional traffic into two separate classes.**

### 3.5 Retransmissions

As the point cloud codec does not support partial decoding, the loss of packets can have a significant impact on the visual quality. Because of this it is important to retransmit any packets that were lost so that the frames are able to be eventually decoded. The proposed architecture uses SACKs to identify which packets were dropped and need to be retransmitted to the client. The data for these dropped packets is retrieved from an in-flight buffer, which stores the data of frames currently in transit. However, this in-flight buffer only has limited capacity, and retains a frame until the client has acknowledged that it has received all packets of that frame.

Due to this limitation, new frames can only be admitted into the buffer when it is not full, leading to frames being dropped when the in-flight buffer is full. However, in contrast to WebRTC, which uses a round-robin buffer that removes frames once filled [21], this approach ensures that once a frame is transmitted, it will ultimately be delivered in its entirety. The size of the in-flight buffer is a critical parameter that must be carefully calibrated based on the specific requirements of the use case. A larger buffer enhances resilience against high round-trip times, but may also increase overall latency in scenarios with significant packet loss.

### 3.6 Packet Queuing

Any packet queuing configuration is allowed, provided it ensures that L4S traffic is isolated from the queuing latency experienced by classic traffic and that L4S packets are marked with CE in anticipation of upcoming congestion [6].

In Section 4, we will consider a queuing setup based on the principles shown in Figure 2. This queue uses the AccECN field to distinguish the L4S traffic from the classic traffic into the two separate classes of the overall queue. Link capacity is distributed between both queues and the aim is to achieve fairness between all flows in each traffic class. The packets within the low-latency queue are marked as CE whenever an early form of congestion is detected. This dual-class setup is designed to ensure the coexistence of L4S traffic with traditional traffic while maintaining low queuing delays for the L4S traffic. Additionally, the utilization of the two classes also does not negatively impact the performance of flows in the traditional queue as the scheduler is configured to prevent starvation of either queue.

### 3.7 Frame Pacing

To mitigate network congestion, an L4S application must minimize burstiness by reducing the number of packets transmitted back to back. In this L4S implementation, we propose to reduce the burstiness by spreading the packets of the frame over a fixed interval. The send time of each packet is calculated using Equation 1.

$$T_{\text{next}} = T_{\text{current}} + \frac{\text{pacing interval}}{\text{frame size}} \times \text{packet size} \tag{1}$$

This pacing mechanism, in conjunction with a fair queue (FQ) scheduler at the sender, ensures that frame data is transmitted gradually rather than in bursts, since the packets are evenly distributed throughout the specified pacing interval. In contrast to GCC, which transmits packets in bursts over intervals of 5 ms, this method exhibits greater fairness toward competing flows. By uniformly spreading the packets across the pacing interval, it maintains consistent burstiness regardless of the available bandwidth.

While the use of pacing introduces additional latency to the pipeline, it effectively prevents sudden spikes in the queuing time of the low-latency queue during frame transmission. This approach ensures that all flows within the low-latency queue can maintain consistent latency and sustain steady throughput, thereby enhancing the overall performance of those flows.

## 4 EVALUATION

This section contains an evaluation of the L4S-based implementation compared to an open-source WebRTC-based volumetric video pipeline [14]. First, we introduce the metrics that will be used in the evaluation. Following this, we describe the experimental setup and the implementation of the L4S compliant queuing setup that was used during the experiments. Then, the results of our experiments in terms of throughput, bandwidth estimation, transport latency, packet loss and visual quality are reported and discussed.

### 4.1 Evaluation Criteria

In our experiments, both the WebRTC-based and L4S-based implementations are evaluated against one or more competing TCP Cubic or TCP Prague flows, initiated by the *iperf* program. TCP Prague is a congestion control algorithm similar to the one in our L4S application, which aims for low, sub 1 ms, queuing delay by reacting to AccECN markings. These competing flows are introduced five seconds after the volumetric application has started. The results of our experiments are reported in two stages. First, we analyze the system's behavior during a startup phase lasting thirty seconds. This analysis allows us to draw significant conclusions regarding initial performance metrics, such as the time required for the congestion controller to converge and the growth rate of the bandwidth estimation. Next, we examine the system's behavior in the stable state, after the first thirty seconds have elapsed. In this stage, we are primarily interested in packet loss, transport latency, and visual quality. It is crucial to evaluate these metrics in the stable state, as the throughput will have stabilized, and the results will no longer be influenced by the initial convergence of the bandwidth estimate. Finally, we compare the total end-to-end latency of both flows.

### 4.2 Experimental Setup

Both the client and L4S server implementations are developed using the C++ programming language. To ensure that packets are not sent in bursts, the server uses a pacing interval of 20 ms and equally paces the packets over this interval using the *SO_TXTIME* socket option and *SCM_TXTIME* packet field to specify the send

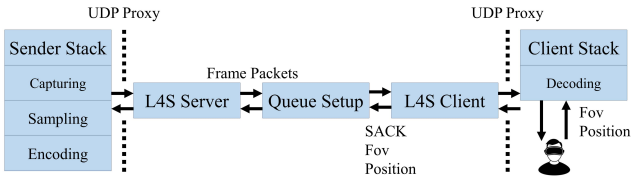



**Figure 3: L4S stack using UDP sockets to transport data between Windows and Linux (L4S applications).**

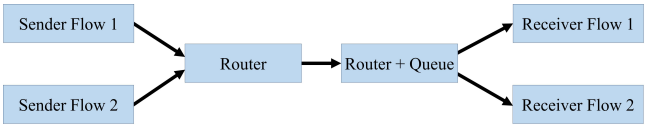



**Figure 4: Experimental setup on the imec Virtual Wall [23].**

time. To ensure that this send time is respected, a FQ is employed at the server side. The capture, encoding, decoding, and rendering components are implemented on a Windows system due to its superior support for VR application. However, given that the current L4S implementation requires capabilities such as setting packet send times and utilizing a sender side FQ, which are only supported by the Linux operating system, the decision was made to implement the networking components on a Linux machine. As shown in Figure 3, a UDP socket proxy is used to transmit data between the Windows-based modules and the L4S server running on Linux, as well as to receive data from the client. This mechanism introduces minimal latency due to the Windows and Linux systems being directly connected via Ethernet on the same network. All experiments utilize a sequence of a single person, captured with an Intel RealSense D455 camera, at a frame rate of 30 FPS with a resolution of 848 × 480 pixels. During the experiments, this precaptured volumetric video is played back at the same speed, using an accurate timer which sleeps for 33 ms between each frame. After background removal, the resulting point clouds contain an average of 125,000 points. The point clouds are then sampled based on the estimated bandwidth and user position by using the sampling method described in Section 3.4. To achieve real-time encoding at 30 FPS, each point cloud is individually encoded using the Draco [16] codec, with position quantization at 11 bits and KD-tree coding enabled. A speed parameter of 7 was selected to maintain an optimal balance between encoding speed and output bitrates. After the sampling and encoding, the resulting bitrates of the sequence vary between 12.5 Mb/s and 75 Mbit/s. In terms of congestion control, the WebRTC-based approach employs GCC, which is the most widely used congestion controller in WebRTC applications. Alternative congestion controllers, such as SCReAM [24], offer a different trade-off between throughput and latency, yielding results that differ from those presented in this section. Both the L4S bandwidth estimator and the GCC algorithm used by WebRTC use a minimum and initial bitrate of 12.5 Mb/s and a maximum rate of 90 Mb/s.

The experimental setup is shown in Figure 4. Throughput and latency experiments are carried out on the imec Virtual Wall, a testbed with a large number of interconnected bare metal nodes [23]. We use several connected nodes with the following hardware specifications; *CPU: Intel Xeon E5520 (2.27GHz), RAM: 12GB DDR3 (1066 MHz), NIC:* 1 Gb/s. Each node uses a patched Ubuntu 20.04 kernel to

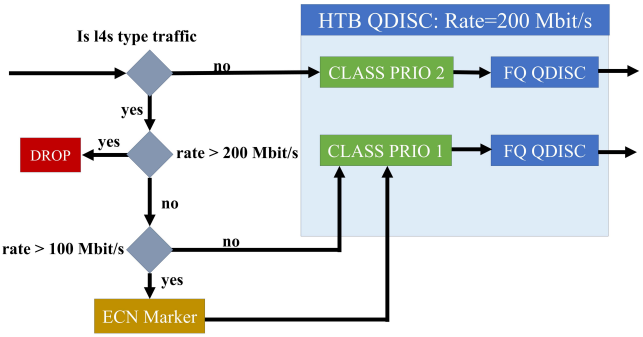



**Figure 5: The detailed implementation of the L4S compliant queuing setup used in the experiments.**

enable support for TCP Prague [28]. This includes the router node, which uses standard Linux features to perform the routing. All experiments are performed with 10 ms of delay added in both directions using traffic control, as well as 200 Mb/s bandwidth restriction on the outgoing interface of the queue.

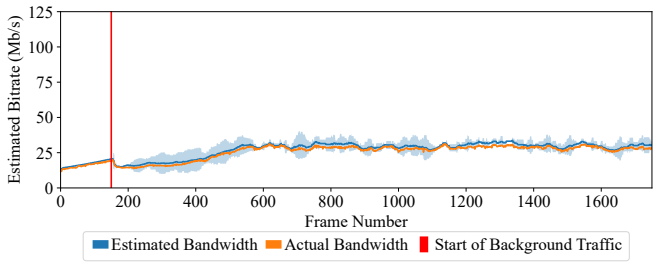
## 4.3 L4S-Compliant Queuing Setup

As described in Section 3.6, the queuing setup is an important part of ensuring that the L4S can properly work. For these experiments the queuing setup shown in Figure 5 was used. The first step of the queue is to separate L4S traffic from traditional flows. This separation is done by checking if the packet has been marked as AccECN-capable transport.

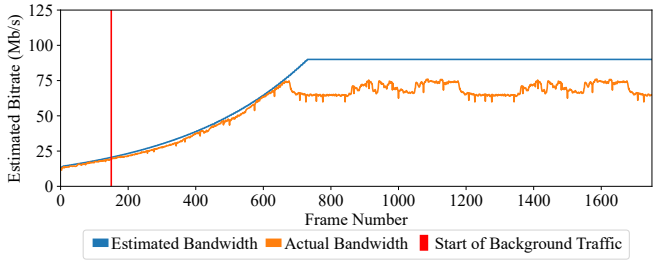
For a traditional flow, no additional processing is performed, and the packet is immediately placed into the queue. However, for L4S traffic, additional steps are taken. First, it is checked if the total rate of all L4S traffic exceeds 200 Mb/s. If this is the case, packets will be dropped immediately, and no AccECN marking will be done. In the next step, an AccECN marker is utilized, setting the AccECN field to CE if the total rate surpasses 100 Mb/s. This marking serves as a notification to the applications that there is a probability that congestion is imminent and that they should lower their send rate to prevent sudden peaks in latency. It is important to emphasize that the packets marked with CE are not dropped and that they proceed through the rest of the queue as normal.

In these experiments, it is assumed that all AccECN-capable flows will respond appropriately to early congestion warnings by reducing their transmission rates. This reduction is intended to prevent actual congestion from occurring. Specifically, by lowering their send rates, the total rate of L4S traffic is kept below 100 Mb/s, ensuring that traditional flows consistently have at least 100 Mb/s of bandwidth available. This mechanism prevents the traditional flows from experiencing starvation due to excessive L4S traffic, which is one of L4S's requirements.

Once packet marking is completed, both L4S and traditional flows are placed into the main hierarchy token bucket (HTB) queue. The L4S traffic is assigned to the higher priority class, giving it preferential access to available bandwidth. However, since packet marking occurs before priority assignment and is triggered at 100 Mb/s, L4S flows are capped at consuming a maximum of 100 Mb/s of the total 200 Mb/s bandwidth. This outcome assumes that all L4S flows appropriately adjust their send rates in response to packet marking to not cross this 100 Mb/s limit.

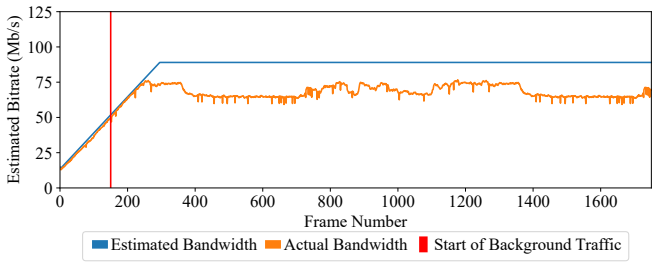



**(a) Three competing TCP Cubic flows**

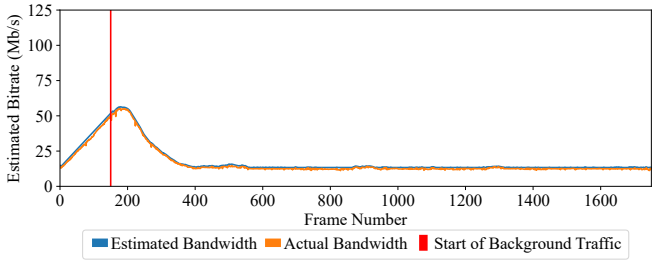



**(b) Three competing TCP Prague flows**

**Figure 6: WebRTC estimated bitrate for different competing flows at 30 FPS.**




**(a) Three competing TCP Cubic flows**




**(b) Three competing TCP Prague flows**

**Figure 7: L4S estimated bitrate for different competing flows at 30 FPS.**

The final component of the queue is a FQ placed at the end of each priority class. This FQ ensures fairness between different flows by operating in a round-robin manner. It cycles through all active flows, taking one or more packets from each before moving to the next. Without this FQ, the default queuing strategy, typically first-in, first-out (FIFO), would be employed. A FIFO approach would disproportionately benefit bursty traffic, as it could dominate the queue and overwhelm other flows, especially those that are more responsive to packet loss.

## 4.4 Achieved Throughput

Figure 6a shows the behavior of the bandwidth estimation and the resulting bitrates for the GCC algorithm used in the WebRTC application, when faced with three competing TCP Cubic flows. This result shows that the maximum bitrate stabilizes at 30 Mb/s once the estimator has converged. This behavior is also present when lowering the number of TCP Cubic flows to two. However, with only one TCP Cubic flow, there is a slight increase in throughput, with the maximum reaching 46 Mb/s. Notably, in this scenario, there are no L4S flows, meaning the entire 200 Mb/s is allocated to traditional traffic. This behavior implies that the expected throughput for WebRTC should be around 75 Mb/s when competing with one or two TCP Cubic flows, if it is assumed that the bandwidth is fairly distributed. The fact that WebRTC fails to achieve this expected throughput suggests that it struggles with the chosen implementation of GCC or queuing setup. In contrast, as shown in Figure 6b, when competing against TCP Prague flows, the WebRTC application is able to reach its maximum throughput. This behavior is because TCP Prague flows receive AccECN markings upon reaching 100 Mb/s of the total 200 Mb/s, prompting them to reduce their transmission rates.

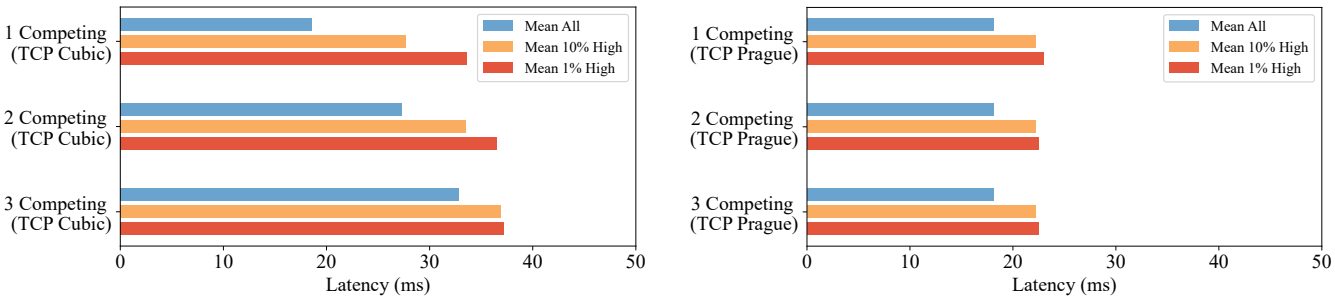
Using the L4S-based application, we observe the results shown in Figure 7. In case of competing TCP Cubic flows, the L4S-based application consistently reaches and maintains its maximum throughput. This outcome is expected, as the L4S application is the only flow in the 100 Mb/s priority queue dedicated to L4S traffic. However, when three TCP Prague flows are introduced, a number of packets receive AccECN markings, causing the throughput of the L4S application to decrease significantly to its minimum value of 12.5 Mb/s. This behavior ensures that the queuing delay is close to 1 ms. Reducing the number of competing flows leads to a higher throughput: with only one TCP Prague flow, the L4S application achieves an average throughput of 58 Mb/s, while with two flows it reaches 15 Mb/s.

## 4.5 Convergence of Bandwidth Estimation

While achieving maximum throughput in a stabilized environment is important, the time it takes to reach this stable state is also a key metric to consider. In these experiments, the initial bitrate for both implementations is set to 12.5 Mb/s, which corresponds to the minimum bitrate necessary to transmit the lowest quality. Starting with this minimal value provides a clear indication of how quickly the estimated bandwidth increases over time until it reaches the maximum estimated bitrate of 90 Mb/s.

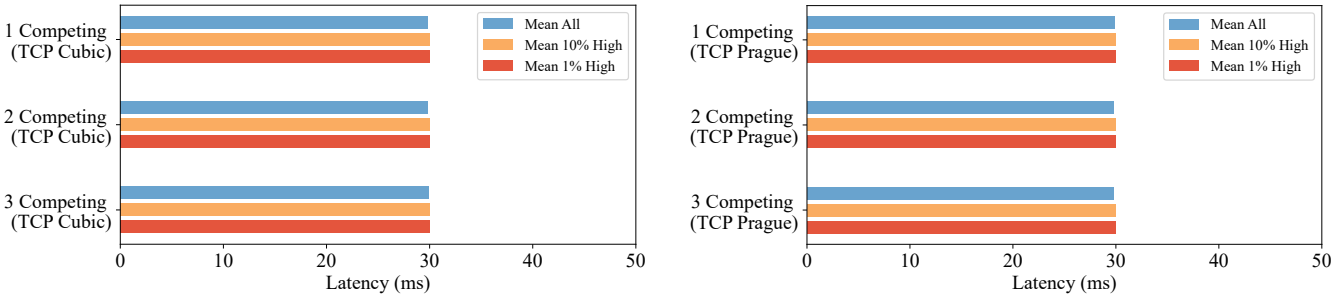
Based on the results presented in the previous section, it is evident that the WebRTC application only achieves its maximum throughput when competing with TCP Prague flows, while the L4S application reaches its maximum throughput when contending with TCP Cubic flows. Due to this fact, it is possible to use these results as an indication of the convergence speed.

As shown in Figure 6b, convergence for the WebRTC application occurs after approximately 22 seconds, which is significantly slower than the L4S-based implementation, where maximum throughput is achieved in just 10 seconds. Although the quality gradually improves during this period, the prolonged convergence of




**(a) Three competing TCP Cubic flows** **(b) Three competing TCP Prague flows**

**Figure 8: WebRTC latency for different competing flows.**




**(a) Three competing TCP Cubic flows** **(b) Three competing TCP Prague flows**

**Figure 9: L4S latency for different competing flows.**

the WebRTC-based implementation time substantially impacts the overall user experience. Furthermore, during periods of temporary congestion, the bitrate will decrease, requiring an extended recovery period to return to optimal levels, which further deteriorates the user's QoE. This slow convergence is observed when looking at the growth behavior of the estimation curve. In the case of L4S, the growth is clearly linear, whereas WebRTC exhibits a more gradual, very slow, non-linear increase.

A potential solution to address the slow convergence is to increase the initial bitrate, which would significantly reduce the time required to reach maximum throughput. However, due to the queuing structure, shown in Figure 5, that utilizes FQ at the end of each priority class, this approach poses certain risks. If a flow would send a sudden burst of data that exceeds its share of bandwidth, the likelihood of packet loss for that flow increases. In contrast, if the burst is lower than the data share, a large amount packet loss will occur for the other flows, as they would not have sufficient time to gradually reduce their transmission rates. Additionally, for L4S traffic, a more gradual increase is more fair towards the other L4S flows as they will not experience a sudden burst of packet markings.

## 4.6 Frame Loss

Due to the early congestion notifications in the L4S implementation, the L4S-based application avoids packet loss by marking packets with AccECN before any are dropped. In contrast, the WebRTC implementation suffers more from packet loss, which it attempts to mitigate through negative acknowledgments (NACKs) to identify and request the lost packets. For this reason, it is critical to focus on the percentage of frames that remain undecodable during transmission for WebRTC. Given that the WebRTC-based implementation uses a NACK retry count of one, each lost packet has only one opportunity for retransmission. Combined with the 1024-packet round-robin buffer for NACKs, this configuration increases the likelihood that certain frames may remain undecodable in favor of achieving a lower latency for the other frames. When competing with TCP Prague flows, the WebRTC implementation suffers no packet loss. However, when faced with TCP Cubic flows, the WebRTC-based implementation experiences non-recoverable packet loss, resulting in approximately 3% of frames being undecodable when contending with three competing TCP Cubic flows.

## 4.7 Peak Transport Latency

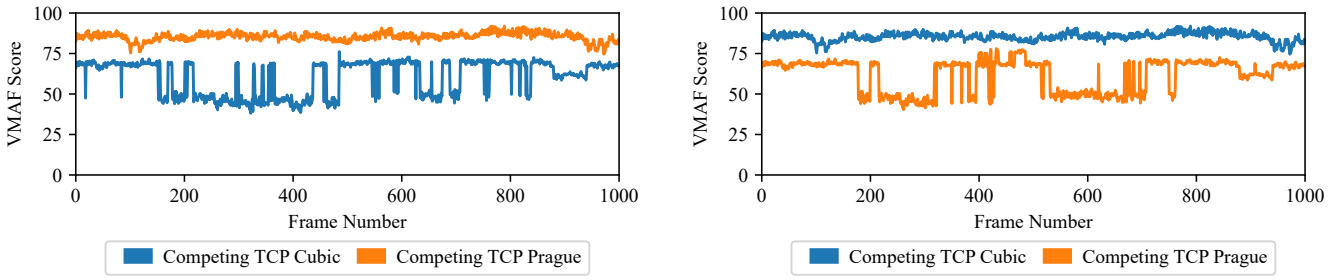
For the latency results, it is important to reiterate that 10 ms of bidirectional latency was introduced and that this latency is also included in the results of this section. As shown in Figure 8, the WebRTC implementation achieves lower latency than the L4S implementation when competing with TCP Prague flows. However, this advantage is only maintained in the presence of fewer than two TCP Cubic flows: when competing with three TCP Cubic flows, the latency for WebRTC surpasses that of the L4S implementation. Furthermore, Figure 8a shows that the latency, which mainly consists of queuing delay, for the WebRTC implementation continues to increase as the number of competing TCP Cubic flows rises, making it more difficult to predict the average latency when the number of competing flows is unknown. In addition to average latency, it is essential to examine peak latency, as these spikes are more noticeable to the end user. Both the WebRTC and L4S applications avoid significant latency spikes. However, in terms of peak latency, the WebRTC implementation only performs better when contending with a single TCP Cubic flow. From two TCP Cubic flows onward, L4S outperforms WebRTC.

**Table 1: VMAF scores for both implementations when competing with different flows, lower intra-sequence scores indicate better quality consistency within a single iteration.**

| Competing flow | TCP Cubic | | | | TCP Prague | | | |
|---|---|---|---|---|---|---|---|---|
| | VMAF | | Intra-sequence VMAF | | VMAF | | Intra-sequence VMAF | |
| | Mean | Std dev | Mean | Std dev | Mean | Std dev | Mean | Std dev |
| WebRTC | 61.98 | 3.26 | 10.53 | 0.83 | 85.39 | 0.00 | 2.57 | 0.00 |
| L4S | 84.97 | 0.33 | 3.16 | 0.52 | 45.17 | 0.04 | 2.41 | 0.34 |

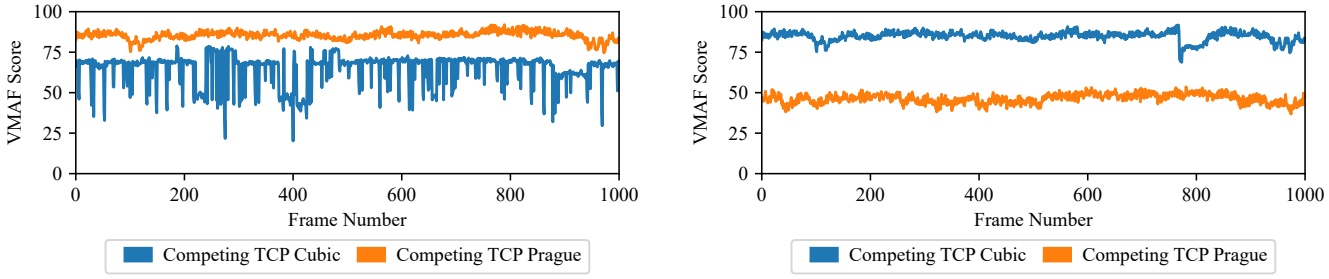
**Table 2: Maximum and minimum VMAF scores for three iterations with a competing three competing TCP Cubic or TCP Prague flows.**

| Competing flow | TCP Cubic | | | | | | Prague | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Iteration | 1 | | 2 | | 3 | | 1 | | 2 | | 3 | |
| | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max |
| WebRTC | 24 | 77 | 11 | 79 | 18 | 79 | 74 | 91 | 74 | 91 | 74 | 91 |
| L4S | 74 | 91 | 74 | 91 | 74 | 91 | 37 | 54 | 37 | 53 | 37 | 64 |




**(a) WebRTC** **(b) L4S**

**Figure 10: WebRTC and L4S VMAF for different competing flows (one concurrent) in the stable state of a single iteration.**




**(a) WebRTC** **(b) L4S**

**Figure 11: WebRTC and L4S VMAF for different competing flows (two concurrent) in the stable state of a single iteration.**

Furthermore, as shown in Figure 9, the latency for L4S remains consistent across all configurations. For L4S, the total latency is primarily determined by the chosen parameters, with 10 ms resulting from the introduced bidirectional latency and an additional 20 ms from the selected pacing interval. This consistency is due to the fact that the L4S implementation effectively shifts the unpredictable latency from queuing to a controlled and predictable pacing latency, allowing it to maintain stable performance regardless of the number of competing flows, compared to the WebRTC-based implementation, where latency continues to increase as more flows are introduced. These results indicate that the proposed L4S approach is successful in maintaining the queuing delay of sub 1 ms.

## 4.8 VMAF Quality

Video Multi-Method Assessment Fusion (VMAF) is a comprehensive video quality metric that merges scores from various video evaluation methods to generate a unified score between 0 and 100. This metric reflects the user-perceived quality more accurately than methods based solely on geometric comparisons of frame differences, since these do not consider the full sequence of frames. In scenarios where frame loss is observed, which in these tests occurs exclusively for WebRTC, the preceding frame is used instead, resulting in a small freeze.

As shown in Table 1, the VMAF scores for WebRTC and the L4S implementation are comparable when competing with a TCP Cubic flow. However, in the presence of a TCP Prague flow, the L4S implementation performs significantly worse, primarily due to its inability to transmit at higher quality.

**Table 3: Latency (ms) introduced by each of the transport-independent pipeline modules.**

| Sender | | Receiver | |
|---|---|---|---|
| Camera | Encoding | Decoding | Rendering |
| 104 ms | 22 ms | 14 ms | 5 ms |

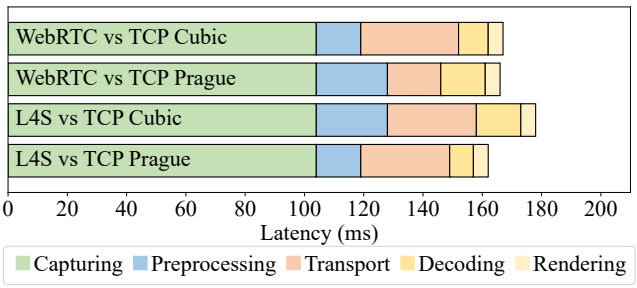



**Figure 12: End-to-end latency for the WebRTC-based and L4S-based implementations when competing with three TCP Cubic or TCP Prague flows.**

While the average VMAF score is an important metric for assessing overall visual quality, it is equally important to examine how the quality evolves during a single iteration. Large fluctuations in quality causes users to become less immersed, which has a significant impact on their experience. As illustrated in Figure 10a, when competing with only one TCP Cubic flow, the WebRTC-based application maintains a relatively stable VMAF score throughout the entire iteration. The minor drops observed can be attributed to variations in frame sizes within the used point cloud sequence. Similar behavior is observed for L4S in Figure 10b. However, for L4S this behavior occurs when competing with a TCP Prague flow instead of a TCP Cubic flow, due to those flows competing over the same bandwidth share.

However, these results change when increasing the number of competing flows to two. As depicted in Figure 11a, when competing with TCP Prague flows, the WebRTC-based implementation exhibits no significant variation, as it continues to utilize the 100 Mb/s share of the available bandwidth. In contrast, when contending with two TCP Cubic flows, the quality fluctuates more noticeably than when competing with only one flow. However, as shown in Figure 11b, the L4S-based implementation does not experience such fluctuations. Instead, it maintains a stable VMAF score, suggesting that although the average score is lower when competing with TCP Prague flows, it achieves a more consistent quality level across the iteration.

These findings are further reflected in Table 1, where the intra-sequence scores reveal significantly greater variance within a single iteration for the WebRTC implementation compared to L4S. Moreover, as shown in Table 2, the WebRTC-based implementation experiences more pronounced drops in visual quality, resulting in a lower minimum VMAF score per iteration than the L4S application. This discrepancy is primarily attributed to frame drops that occur exclusively in the WebRTC implementation.

## 4.9 End-to-End Latency Breakdown

Most components of the pipeline are independent of the adopted transport protocol. Table 3 shows the average latency for each of these components, with the camera contributing the most to the transport-independent end-to-end latency. However, when considering the impact of the transport protocol along with three competing flows, the results in Figure 12 reveal that both the WebRTC-based and L4S-based implementations achieve similar end-to-end latencies. The slight differences in encoding and decoding times can be attributed to higher throughput leading to longer processing times. Nevertheless, as noted in Section 4.7, the L4S implementation exhibits significantly more consistent latency compared to WebRTC, with the key advantage being that L4S latency remains unaffected by the number of competing flows.

# 5 CONCLUSION AND FUTURE WORK

In this paper, we propose a low latency, low loss and scalable throughput (L4S)-based architecture for volumetric video conferencing using point clouds. We perform a comprehensive comparison of the proposed architecture against an open-source, web real-time communication (WebRTC)-based volumetric video delivery pipeline [14], which utilizes WebRTC to enable real-time volumetric conferencing. Our L4S-based implementation demonstrates a more stable transport latency by shifting unpredictable queuing delays to a predictable latency through packet pacing. Consequently, the transport latency of the proposed L4S architecture is determined by the pacing interval and any fixed network latency, whereas the WebRTC-based implementation experiences increased latency as the number of competing Cubic flows grows. Moreover, the L4S implementation leverages accurate explicit congestion notification (AccECN) markings to receive early congestion warnings, adjusting its rate accordingly, which prevents most packet loss. In contrast, the WebRTC implementation suffers from packet loss, with 3% of frames becoming undecodable due to missing packets and the failure of negative acknowledgment (NACK)-based retransmission. Additionally, the L4S bandwidth estimator converges from 12.5 Mb/s to 90 Mb/s within 10 seconds, representing a 45% faster convergence rate compared to the Google congestion control (GCC) estimator used in the WebRTC-based implementation. However, implementing the L4S-based approach requires bottleneck routers to have a fair queuing setup that supports AccECN marking, which may be challenging for older or legacy networks.

In future work, we plan to extend the implementation based on L4S to support bidirectional communication. This extension will enable the implementation of a many-to-many communication scenario, providing an opportunity to evaluate the performance of L4S in environments with multiple senders and receivers. Such a scenario will yield valuable insights into the scalability and efficiency of L4S in more complex, real-world conferencing setups. Furthermore, since both the WebRTC and L4S-based implementations are designed to be content-independent, extending the system to support various types of volumetric video, such as mesh-based formats, would greatly enhance the modularity of the pipeline. This flexibility is further bolstered by the use of the simple sampling quality adaptation method and the Draco codec, which both support mesh-based encoding.

# REFERENCES


[1] H. Afzal, D. Aouada, B. Mirbach, and B. Ottersten. 2018. Full 3D reconstruction of non-rigidly deforming objects. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14, 1s.

[2] S. J. Ahn, L. Levy, A. Eden, A. S. Won, B. MacIntyre, and K. Johnsen. 2021. IEEEVR2020: exploring the first steps toward standalone virtual conferences. *Frontiers in Virtual Reality*, 2, 648575.

[3] D. S. Alexiadis, D. Zarpalas, and P. Daras. 2012. Real-time, full 3-D reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*, 15, 2, 339–358.

[4] M. Bassier, M. Vergauwen, and F. Poux. 2020. Point cloud vs. mesh features for building interior classification. *Remote Sensing*, 12, 14, 2224.

[5] N. Bouzakaria, C. Concolato, and J. Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*. IEEE, 92–97.

[6] B. Briscoe, K. De Schepper, M. Bagnulo, and G. White. 2023. Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. RFC 9330. (Jan. 2023). DOI: 10.17487/RFC9330.

[7] B. Briscoe, K. De Schepper, O. Tilmans, M. Kühlewind, J. Misund, O. Albisser, and A. S. Ahmed. 2019. Implementing the'prague requirements' for low latency low loss scalable throughput (L4S). *Netdev 0x13*.

[8] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. Leung, and C.-H. Hsu. 2016. A survey on cloud gaming: future of computer games. *IEEE Access*, 4, 7605–7620.

[9] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. 2016. BBR: congestion-based congestion control: measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14, 5, 20–53.

[10] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. 2016. Analysis and design of the Google congestion control for web real-time communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems*, 1–12.

[11] 2024. Cwi point cloud codec. (2024). https://github.com/cwi-dis/cwipc_codec.

[12] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou. 2017. 8i voxelized full bodies-a voxelized point cloud dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059 / WG1M74006*, 7, 8.

[13] W. De Bruijn and E. Dumazet. 2018. Optimizing UDP for content delivery: GSO, pacing and zerocopy. In *Linux Plumbers Conference*.

[14] M. De Fré, J. van der Hooft, T. Wauters, and F. De Turck. 2024. Scalable MDC-based volumetric video delivery for real-time one-to-many WebRTC conferencing. In *Proceedings of the 15th ACM Multimedia Systems Conference*, 121–131.

[15] K. De Schepper, O. Albisser, O. Tilmans, and B. Briscoe. 2022. Dual queue coupled AQM: deployable very low queuing delay for all. *arXiv:2209.01078*.

[16] [SW] Google, Draco 2023. URL: https://google.github.io/draco/.

[17] W. Feng, D. Kandlur, D. Saha, and K. Shin. 1997. Techniques for eliminating packet loss in congested TCP/IP networks. Tech. rep. Citeseer.

[18] S. Garg, A. Srivastava, M. Glencross, and O. Sharma. 2022. A study of the effects of network latency on visual task performance in video conferencing. In *CHI Conference on human factors in computing systems extended abstracts*, 1–7.

[19] P. Graff, X. Marchal, T. Cholez, B. Mathieu, S. Tuffin, and O. Festor. 2024. Improving cloud gaming traffic QoS: a comparison between class-based queuing policy and L4S. In *Network Traffic Measurement and Analysis Conference (TMA 2024)*. IEEE, 10.

[20] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. 2020. An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC). *APSIPA Transactions on Signal and Information Processing*, 9.

[21] S. Holmer, M. Shemer, and M. Paniconi. 2013. Handling packet loss in WebRTC. In *2013 IEEE International Conference on Image Processing*. IEEE, 1860–1864.

[22] X. Hu, C. Wang, Y. Pan, Y. Liu, Y. Wang, Y. Liu, L. Zhang, and S. Shirmohammadi. 2021. 4DLFVD: a 4D light field video dataset. In *Proceedings of the 12th ACM Multimedia Systems Conference*, 287–292.

[23] [SW], imec Virtual Wall 2023. URL: https://doc.ilabt.imec.be/ilabt/virtualwall/.

[24] I. Johansson. 2014. Self-clocked rate adaptation for conversational video in LTE. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, 51–56.

[25] A. B. Johnston and D. C. Burnett. 2012. *WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web*. Digital Codex LLC.

[26] B. Kenwright. 2019. Virtual reality: where have we been? where are we now? and where are we going?

[27] R. Neff and A. Zakhor. 1997. Very low bit-rate video coding based on matching pursuits. *IEEE Transactions on circuits and systems for video technology*, 7, 1, 158–171.

[28] 2024. Patched l4s linux kernel. (2024). https://github.com/L4STeam/linux.

[29] M. P. Sarma. 2013. Performance measurement of TCP and UDP using different queuing algorithm in high speed local area network. *International Journal of Future Computer and Communication*, 2, 6, 682.

[30] D. Shan and F. Ren. 2018. Ecn marking with micro-burst traffic: problem, analysis, and improvement. *IEEE/ACM Transactions on Networking*, 26, 4, 1533–1546.

[31] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. 2020. Latency and cybersickness: impact, causes, and measures. a review. *Frontiers in Virtual Reality*, 1, 582204.

[32] J. Tideström. 2019. Investigation into low latency live video streaming performance of WebRTC. (2019).

[33] J. van der Hooft, H. Amirpour, M. T. Vega, Y. Sanchez, R. Schatz, T. Schierl, and C. Timmerer. 2023. A tutorial on immersive video delivery: from omnidirectional video to holography. *IEEE Communications Surveys & Tutorials*.

[34] J. van der Hooft, T. Wauters, F. De Turck, C. Timmerer, and H. Hellwagner. 2019. Towards 6DoF HTTP adaptive streaming through point cloud compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, 2405–2413.

[35] I. Viola, J. Jansen, S. Subramanyam, I. Reimat, and P. Cesar. 2023. Vr2gather: a collaborative, social virtual reality system for adaptive, multiparty real-time communication. *IEEE MultiMedia*, 30, 2, 48–59.

[36] D. Vucic and L. Skorin-Kapov. 2019. The impact of packet loss and google congestion control on QoE for WebRTC-based mobile multiparty audiovisual telemeetings. In *MultiMedia Modeling: 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8–11, 2019, Proceedings, Part I 25*. Springer, 459–470.

[37] Wowza. 2021. 2021 Video Streaming Latency Report. Tech. rep. Wowza.

[38] M. Yang, Z. Luo, M. Hu, M. Chen, and D. Wu. 2023. A comparative measurement study of point cloud-based volumetric video codecs. *IEEE Transactions on Broadcasting*.