Towards Efficient Transport for Real-Time Immersive Applications over Hybrid Networks

Casper Haems, Matthias De Fré, Tim Wauters, Filip De Turck
Department of Information Technology
Ghent University - imec
Ghent, Belgium
casper.haems@ugent.be, matthias.defre@ugent.be, tim.wauters@ugent.be, filip.deturck@ugent.be

***Abstract*—Immersive telepresence demands high data rates and low latency, yet no single commercial data path reliably meets these needs. Fine-grained content selection also remains underdeveloped. This work proposes a hybrid, multi-path delivery framework combining broadcast and unicast into a single service. A lightweight base scene is broadcast via File Delivery over Unidirectional Transport (FLUTE), ensuring no viewer ever sees a fully blank scene, while viewer-specific enhancements are steered over unicast. An open-source testbed is released to investigate the impact of network impairments, instrument common protocols, and enable reproducible experiments. On high-quality volumetric video (up to 100k points per frame at 30 frames per second), the hybrid design (i) keeps latency below 40 ms while scaling quality with unicast bandwidth, (ii) reduces server and network load compared to pure unicast, and (iii) masks typical wireless loss patterns with only 15% Forward Error Correction (FEC) overhead. These findings show that treating broadcast and unicast as complementary channels is crucial for scalable Extended Reality (XR) services.**

***Index Terms*—Volumetric video, hybrid broadcast-unicast, multi-path transport, real-time streaming, immersive media**

I. Introduction

Immersive communication, ranging from volumetric telepresence and free-viewpoint sports to collaborative Extended Reality (XR), aims to stream dynamic point cloud video to head-mounted displays with low latency and high quality. While 5G/6G roadmaps envision such holographic services with slicing and virtualization, real-world deployments remain limited by bandwidth-hungry, latency-sensitive streams traversing lossy, congested networks [1], [2].

Existing solutions each address only part of the problem: Dynamic Adaptive Streaming over HTTP (DASH) and HTTP Live Streaming (HLS) offer adaptivity but incur high latency; Web Real-Time Communication (WebRTC) achieves sub-200 ms delay but still consumes substantial bandwidth; File Delivery over Unidirectional Transport (FLUTE) broadcasts efficiently at scale but lacks personalization [3]–[7]. Work on hybrid broadcast-unicast architectures shows promise for 2D video but has yet to be extended to volumetric media. This work closes the gap with a hybrid, multi-path transport that broadcasts a base scene over FLUTE and unicasts viewer-specific enhancements via DASH. The goal is to maintain a never-blank scene in real time while scaling to many viewers and adapting quality to available bandwidth. This paper contributes: (i) an open-source testbed to evaluate real-time volumetric streams over hybrid paths[1], (ii) a unified metric suite capturing latency, throughput, quality, and frame drops, (iii) a head-to-head evaluation of WebRTC, WebSocket, DASH, and FLUTE under network impairments, (iv) and a hybrid, multi-path design that balances scalability and personalization. Figure 1 summarizes the concept: a base description is broadcast to all, while enhancements are delivered on demand over unicast. The following sections describe related work, system architecture, metrics and methodology, experimental results, conclusion, and future work.

II. Related Work

Three lines of research that converge on the goal of a hybrid, multi-path transport for real-time immersive media are surveyed in this section: streaming protocols, immersive-media delivery systems, and hybrid unicast-broadcast architectures.

A. Protocols for Streaming

Conventional HTTP Adaptive Streaming (HAS) protocols (DASH, HLS) divide video into multi-second segments and rely on Transmission Control Protocol (TCP), incurring live latencies above 5 s [3]–[5], [8]. Even low-latency variants (Low Latency DASH (LL-DASH), Low-Latency HLS (LL-HLS)) remain above 1 s due to TCP setup, segment buffering, and head-of-line blocking [9], [10]. Additionally, WebSocket and hypertext transfer protocol (HTTP)/2 Push stream media over persistent TCP connections to reduce latency [11], [12]. User Datagram Protocol (UDP)-based protocols address TCP's limitations. Secure Reliable Transport adds Automatic Repeat Request for sub-second live streaming in professional setups, though web browsers do not support it [13]. WebRTC is the de-facto choice for sub-second delivery, combining Real-time Transport Protocol framing with application-level congestion control [14]. Although originally designed for peer-to-peer, server-side Selective Forwarding Units architectures enable scaling to many clients [15]. Finally, QUIC and the emerging Media over QUIC Transport (MOQT) combine UDP's low delay with multiplexing and publish/subscribe abstractions [16], [17]. Early prototypes achieve WebRTC-like latency, but specifications and support remain nascent [10].

[1]https://github.com/idlab-discover/Multi-path-XR

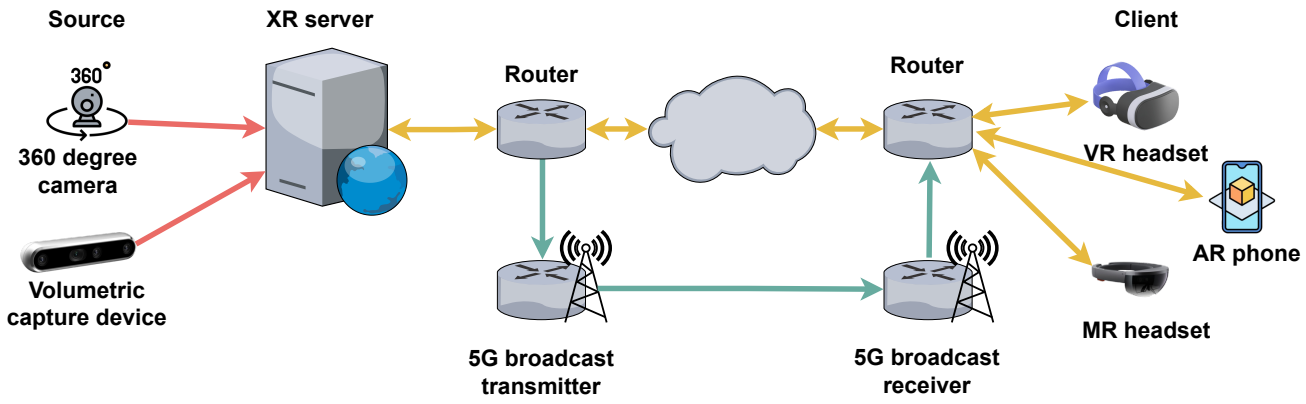

Fig. 1. End-to-end hybrid delivery concept for real-time volumetric media. Content is captured either by a 360° camera or by a volumetric capture device capable of producing point-cloud video. After processing on an XR server, the media are distributed over two complementary paths: the conventional Internet Protocol broadband network and a 5G broadcast link. The streamed scene can then be consumed on Virtual Reality headsets, Mixed Reality headsets and phones with Augmented Reality functionality.

B. Immersive Media Delivery Systems

Real-time volumetric streaming requires much higher bandwidth than 2D video while meeting similar latency constraints. Early work adapted 2D streaming, van der Hooft et al. encoded each object at multiple qualities and delivered them with DASH-based six degrees-of-freedom (6DoF) streaming, which selects quality based on user position, field of view (FOV) and bandwidth [18]. However DASH incurs higher delay than UDP-centric transports, motivating other designs. VR2Gather enables multi-party telepresence with multiple TCP-based transports [19]. De Fré et al. improved upon this concept with an open-source conferencing platform using Draco-encoded point clouds over WebRTC, achieving ∼160 ms end-to-end latency while using Multiple Description Coding (MDC) [6], [20]. LiveVV separates background and foreground, applies point decimation and volumetric-aware adaptive bitrate streaming (ABR), sustaining 24 frames per second (FPS) with ≤350 ms latency [21]. LL-Sparse predicts the FOV to pre-fetch only visible data [22]. New datasets such as UVG-VPC and MazeLab enable realistic evaluation of such techniques [23], [24]. Together, these works highlight how protocol choice, content optimization, and predictive delivery enable 6DoF streaming. However, none exploit multiple delivery paths concurrently.

C. Hybrid Unicast-Broadcast Architectures

Hybrid streaming reduces per-user bandwidth by offloading common data to a broadcast channel, while reserving unicast for personalized refinements. For example, a scene's static background or base layer can be broadcast, while dynamic foregrounds or enhancements use unicast. Haems et al. integrate and demonstrate 5G terrestrial broadcast with over-the-top, 2D linear video streaming, achieving sub-second live latency and significantly reducing unicast traffic via FLUTE with Forward Error Correction (FEC) [7], [25]. Advanced Television Systems Committee 3.0 and 3rd Generation Partnership Project adopt comparable schemes [26], [27]. These studies show that hybrid architectures effectively reduce bandwidth and latency for 2D shared content. However, they stop short of volumetric media, and no open, reproducible framework exists for hybrid volumetric streaming, a gap this work addresses.

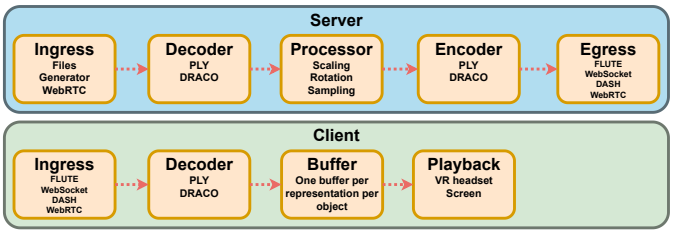

Fig. 2. Modular pipeline for real-time volumetric media.

III. System Architecture

Figure 2 depicts the modular pipeline, with mostly symmetrical server (blue) and client (green) paths. Both consist of interchangeable blocks for codecs, processing, and transport. If needed, the decoder, processor and encoder of the server can be skipped altogether.

a) Ingress and Decoder: The server ingests either pre-encoded datasets (from files) or live streams (WebRTC) and decodes them into point clouds. Both server and client use the same decoder, supporting Polygon File Format and Draco.

b) Processor: This optional block scales, rotates, downsamples, or combines point clouds. It can also split the stream into multiple disjoint point clouds for ABR or MDC.

c) Encoder: Processed frames are encoded with a selectable codec before transmission.

d) Egress (Server): Encoded frames are multiplexed across one or more protocols: (i) FLUTE broadcasts with block-level FEC over UDP using IP multicast, (ii) WebSocket (Socket.IO) pushes content over TCP, (iii) DASH, as the conventional ABR approach, pulls frames as mp4 segments using HTTP, and (iv) WebRTC provides a unicast UDP transport as a real-time approach. The design is protocol-agnostic, and

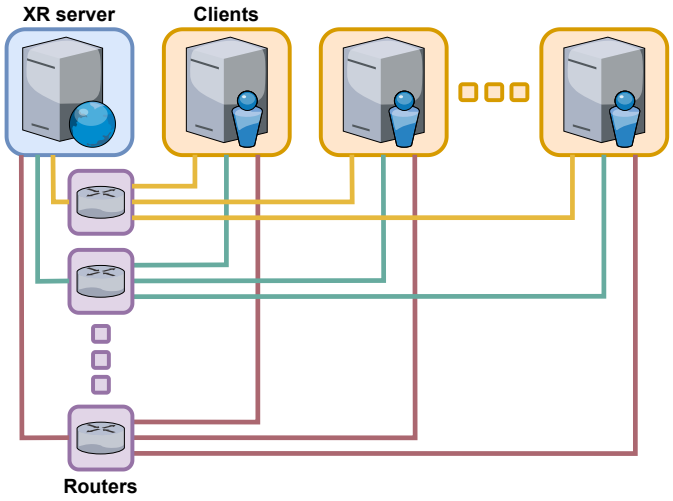

Fig. 3. Network topology: each box is a node. The number of servers, clients, and routers is configurable.

future work could add MOQT for fine-grained content and quality prioritization.

e) Ingress (Client): Clients mirror the server egresses: they open WebSocket, WebRTC, FLUTE, or DASH sessions and feed decoded frames into a minimal playback buffer. Unlike conventional ABR, this DASH ingress uses only an Exponentially Weighted Moving Average-based estimator without buffer-based smoothing for reduced latency.

This path-aware implementation, written in Rust, supports hybrid experiments by allowing any transport combination at runtime. Multithreading enables frame $n{+}1$ to decode while decoding of frame n is still in progress, preventing congestion on one path from blocking others.

IV. Evaluation Metrics and Methodology

This section outlines the experimental testbed, followed by the metrics used to evaluate each transport.

A. Experimental Setup

All trials are conducted in a Mininet environment on an Intel® Core™ i7-1365U with 32 GB RAM running Ubuntu 24.04 LTS [28]. The emulated network topology is shown in Figure 3, where each box represents one node on the Mininet topology. Linux `tc` enforces bandwidth, delay, and packet-loss profiles individually per link displayed on the right side of the routers. Each scenario runs for ten minutes, with the first and last minute discarded to exclude start-up and tear-down artifacts, leaving eight minutes of steady-state measurements. The server streams point clouds as frames in a loop taken from the `longdress` sequence from the MPEG PCC dataset [29]. Table I lists the rates for the downsampled versions of this sequence. All sequences are encoded with the Draco codec. The three smaller versions contain disjoint point sets of the 100k version, enabling both traditional ABR and MDC experiments. For example, broadcasting 15k points as a base description while unicasting the 25k description combines into a 40k reconstructed description. Loss of packets on a single path therefore degrades quality gracefully rather than interrupting playback.

TABLE I
Bitrate of longdress at different qualities (Draco-encoded, 30 fps).

Number of points	Visual quality	Mbps
15k	Low	16.57 ± 0.05
25k	Medium-low	26.38 ± 0.09
60k	Medium	58.17 ± 0.23
100k	High	91.96 ± 0.40

TABLE II
Throughput and latency at 200 Mbps bandwidth cap for 100k-point stream.

Protocol	Throughput (Mbps)	Latency (ms)
DASH	118.58	22.33
WebSocket	117.83	21.39
WebRTC	100.70	17.18
FLUTE	97.66	17.99

B. Metrics

Running on a single host enables clock synchronization for accurate latency measurements, at the cost of increased computational load. Latency is measured as the time between the moment a frame is triggered to be sent using the selected egress at the server and the moment the frame is fully received at the client. This latency includes packetization, FEC, transport and depacketization and excludes the time for preprocessing, encoding and decoding the point cloud. As such, the metric measures the full latency of the transport protocol. In addition, the frame rate, point cloud encoding and decoding time, CPU and memory usage, the number of points received and other relevant metrics, such as the throughput per link, are measured.

V. Experimental Results

The evaluation proceeds in two stages. Section V-A quantifies the behavior of transports, establishing reference points for latency, bandwidth efficiency, and robustness. Section V-B then activates multiple egresses simultaneously to demonstrate the benefits and trade-offs of the proposed hybrid design.

A. Single-Path Baselines

a) Transport efficiency under bandwidth ceilings: The baseline scenario streams the `longdress` sequence in a loop at 30 FPS with a fixed quality of 100k points per frame encoded using Draco. This source has a bitrate of 91.96 Mbps. The links between server and the client are throttled to 200 Mbps. Table II summarizes the measured throughput and latency per protocol. UDP-based protocols (WebRTC, FLUTE) incur only ∼8% overhead relative to the source bitrate, while TCP-based protocols (DASH, WebSocket) incur ∼25% more. When decreasing the bandwidth until below the source rate, FPS degrades gradually for TCP (due to congestion control)

TABLE III
LATENCY OF FLUTE WITH DIFFERENT FEC SCHEMES.

FEC scheme	Latency (ms)
No FEC	17.99
Raptor	19.36
RaptorQ	18.86
Reed-Solomon GF(2^8)	25.47
Reed-Solomon GF(2^8) (Under Specified)	24.30

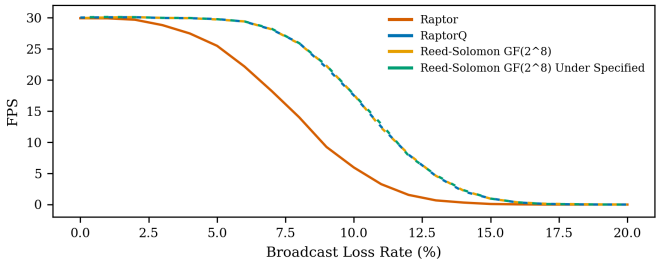

Fig. 4. Frame-rate resilience of FLUTE with 15% FEC under increasing packet-loss rates. Decoding success rate is non-linear and can be inferred from the frame-rate curve.

but drops abruptly for UDP once the path saturates due to the lack of congestion control. It is thus recommended to enable a congestion control scheme for WebRTC. Latency remains low and stable for all tested protocols when bandwidth suffices. Among the protocols, only WebSocket implements backpressure, keeping its latency more stable under congestion. FLUTE throughput remains constant under multiple clients, unlike unicast protocols which scale linearly. Likewise, server-side processing demands remain constant for FLUTE, but increase with the number of receivers for unicast-based approaches.

b) Loss resilience with forward-error correction: Adding 15% FEC overhead to the 100k FLUTE stream, increases throughput from 97.66 Mbps to 119.94 Mbps, but significantly improves robustness to packet loss. Table III shows the latency for four common schemes [30]–[33]. RaptorQ achieved the best trade-off between overhead and latency while tolerating up to 15% loss, as shown in Figure 4.

B. Hybrid Multi-Path Evaluation

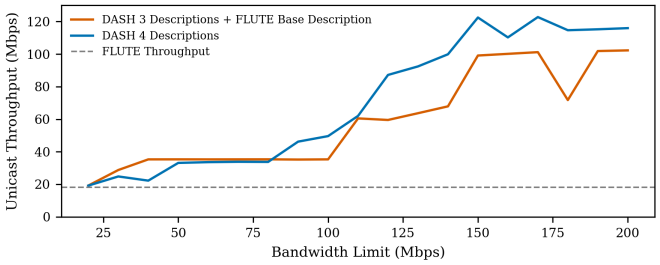

Fig. 5. Average unicast throughput of the stream in the setup as the unicast bandwidth cap is increased. The throughput used by FLUTE for transmitting the baseline quality with a bandwidth cap of 200Mbps is shown for reference.

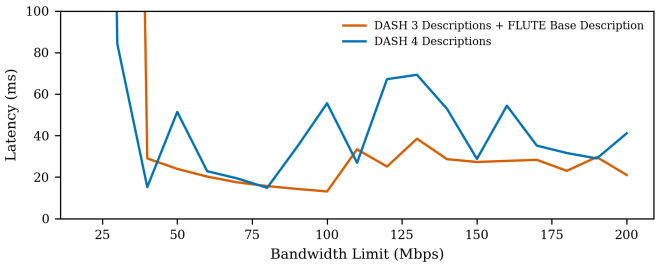

Fig. 6. Average latency of the setup for DASH as the unicast bandwidth cap is increased. Latency depends on the selected quality representation, which depends on the bandwidth cap.

A hybrid setup is evaluated combining broadcast and unicast paths. The scene is split into 15k (base), 25k, and 60k qualities, with an additional 85k refinement quality (25k+60k). The 15k base is broadcast via FLUTE to ensure all clients receive at least a minimal-quality scene, while higher qualities (25k, 60k, 85k) are delivered adaptively over unicast using DASH. This aims to reduce per-client network load while enhancing quality by combining broadcast and unicast streams. This scenario is compared with an adaptive, unicast-only stream using DASH that can choose between four qualities as well (15k, 25k, 60k, 100k). Figure 5 shows unicast throughput as the unicast bandwidth cap increases, with the broadcast path streaming at 18.09 Mbps. Hybrid delivery reduces unicast load by offloading the base layer to broadcast, especially at higher bandwidth capacities where DASH does not need to stream the 100k quality, reducing the unicast throughput by ∼18 Mbps. As mentioned in Section III, the client runs a latency-first ABR strategy, which lacks additional heuristics such as playback buffer-based ABR smoothing. Figure 6 shows that latency remains below 40 ms, though the simplistic ABR implementation occasionally oscillates between qualities due to its aggressive, latency-first adaptation. This is also visible in Figure 5 as it sometimes tries to stream a higher quality than the available bandwidth allows. Compared to the 100k quality-only stream in Table II, latency is more variable because it depends on which quality DASH fetches. Using a larger playback buffer would allow more ABR heuristics and stabilize the quality but at the cost of higher playback delay. In conclusion, the hybrid adaptation meets its targets: it keeps latency below 40 ms, and always delivers at least the 15k baseline while upgrading overall quality, and significantly reduces overall network and server load.

VI. CONCLUSION

This paper has presented the first open, reproducible framework for real-time immersive applications over hybrid networks, evaluating four transport approaches: WebRTC, WebSocket, DASH, and FLUTE. The contributions are threefold: (i) an open-source testbed for point cloud video streams with controllable network impairments, (ii) a unified metric suite covering latency, throughput, and other metrics, and (iii) a hybrid sender/receiver that broadcasts a base description while

unicasting enhancements. Results reveal clear trade-offs: UDP-based WebRTC and FLUTE add under 10% bandwidth overhead and sustain sub-30 ms latency when link capacity meets the source rate, whereas TCP-based WebSocket and DASH incur ∼25% overhead. A modest 15% RaptorQ FEC overhead masks random losses on the broadcast path without noticeable delay. The hybrid approach ensures the scene is never blank, scales quality with available unicast bandwidth, and reduces the linear traffic growth of pure unicast by offloading shared content to broadcast. A carefully orchestrated hybrid architecture can meet the dual demands of real-time latency and multi-gigabit volumetric streaming, paving the way for practical XR services on forthcoming 5G/6G deployments. Taken together, these findings demonstrate that hybrid multi-path orchestration is essential for scalable immersive communication. By treating broadcast and unicast as complementary, the system meets both the scalability and user experience demands.

VII. Future Work

Future work will extend the current frame-level prototype to object-level streaming, where individual scene elements (e.g., background, avatars, props) are assigned priorities and mapped to appropriate paths: high-saliency objects over unicast for personalized quality, and low-saliency ones over best-effort broadcast. This finer granularity would enable better ABR decisions and allow protocols such as MOQT to enforce per-object priorities, reducing bandwidth waste and improving perceptual quality. Moreover, the Mininet-based evaluation is limited to static topologies, tens of flows and single-host computation. Scenarios with mobility, uplink contention, and cross-traffic remain unexplored. Thanks to its open-source nature, the framework can be replayed on larger testbeds or public clouds. Future work will port it to larger-scale platforms and deploy it in 5G/6G mobility trials to assess real-world performance under dynamic conditions.

Acknowledgment

This work has been funded by the European Union (SPIRIT project, Grant Agreement 101070672, https://www.spirit-project.eu/).

References

[1] Z. Liu et al., "Point cloud video streaming: Challenges and solutions," *IEEE Network*, vol. 35, no. 5, pp. 202–209, 2021.

[2] R. S. Kalan, S. Clayman, and M. Sayit, "vDANE: Using virtualization for improving video quality with server and network assisted dash," *Int. J. Netw. Manag.*, vol. 32, no. 5, p. e2209, 2022.

[3] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[4] ISO, "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats," Tech. Rep., 2022. [Online]. Available: https://www.iso.org/standard/83314.html

[5] R. Pantos and W. May, "HTTP Live Streaming," RFC 8216, 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8216

[6] M. De Fré et al., "Scalable MDC-based volumetric video delivery for real-time one-to-many WebRTC conferencing," in *Proc. ACM MMSys '24*, 2024, pp. 121–131.

[7] C. Haems et al., "Enabling adaptive and reliable video delivery over hybrid unicast/broadcast networks," in *Proc. ACM NOSSDAV '24*, 2024, pp. 29–35.

[8] J. van der Hooft et al., "A tutorial on immersive video delivery: From omnidirectional video to holography," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1336–1375, 2023.

[9] ISO, "Information technology - Multimedia application format (MPEG-A) - Part 19: Common media application format (CMAF) for segmented media," Tech. Rep., 2024. [Online]. Available: https://www.iso.org/standard/85623.html

[10] Z. Gurel et al., "Media-over-QUIC transport vs. low-latency dash: a deathmatch testbed," in *Proc. ACM MMSys '24*, 2024, pp. 448–452.

[11] B.M. Tabari et al., "Low latency live video streaming on android devices using web-socket," in *ICCCNT '17*, 2017, pp. 1–6.

[12] J. van der Hooft et al., "An HTTP/2 push-based approach for low-latency live streaming with super-short segments," *JNSM*, vol. 26, no. 1, pp. 51–78, 2018.

[13] M. Sharabayko et al., "The SRT Protocol," IETF, Internet-Draft draft-sharabayko-srt-01, 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-sharabayko-srt/01/

[14] B. Jansen et al., "Performance evaluation of webrtc-based video conferencing," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, pp. 56–68, 2018.

[15] H. T. Alvestrand, "Overview: Real-Time Protocols for Browser-Based Applications," RFC 8825, 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8825

[16] L. Curley et al., "Media over QUIC Transport," 2024. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-moq-transport/03/

[17] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[18] J. van der Hooft et al., "Towards 6DoF HTTP adaptive streaming through point cloud compression," in *Proc. ACM MM '19*, 2019, pp. 2405–2413.

[19] I. Viola et al., "Vr2gather: A collaborative, social virtual reality system for adaptive, multiparty real-time communication," *IEEE MultiMedia*, vol. 30, no. 2, pp. 48–59, 2023.

[20] M. De Fré et al., "Demonstrating adaptive many-to-many immersive teleconferencing for volumetric video," in *Proc. ACM MMSys '24*, 2024, pp. 453–458.

[21] K. Hu et al., "Livevv: Human-centered live volumetric video streaming system," *IEEE Internet of Things Journal*, pp. 1–1, 2025.

[22] J. Ouellette and A. Bentaleb, "LL-Sparse: Low-latency 6-dof field of view prediction," in *Proc. ACM MMSys '25*, 2025, pp. 57–67.

[23] G. Gautier et al., "Uvg-vpc: Voxelized point cloud dataset for visual volumetric video-based coding," in *QoMEX '23*, 2023, pp. 244–247.

[24] J. Ouellette, J. S. Sidhu, and A. Bentaleb, "MazeLab: A large-scale dynamic volumetric point cloud video dataset with user behavior traces," in *Proc. ACM MMSys '25*, 2025, pp. 298–304.

[25] C. Haems et al., "Real-time demonstration of low-latency video delivery via hybrid unicast-broadcast networks," in *CNSM '24*, 2024, pp. 1–3.

[26] J. Lee et al., "Ip-based cooperative services using atsc 3.0 broadcast and broadband," *IEEE Trans. on Broadcasting*, vol. 66, no. 2, pp. 440–448, 2020.

[27] D. Silhavy et al., "3GPP Rel-17 5G media streaming and 5G broadcast powered by 5G-MAG reference tools," in *Proc. ACM MHV '23*, 2023, pp. 85–90.

[28] N. Handigol et al., "Reproducible network experiments using container-based emulation," in *Proc. ACM CoNEXT '12*, 2012, pp. 253–264.

[29] E. d'Eon et al., "8i Voxelized Full Bodies - A Voxelized Point Cloud Dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006, 2017, available at https://plenodb.jpeg.org/pc/8ilabs/.

[30] A. Shokrollahi et al., "Raptor Forward Error Correction Scheme for Object Delivery," RFC 5053, 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5053

[31] L. Minder et al., "RaptorQ Forward Error Correction Scheme for Object Delivery," RFC 6330, 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6330

[32] V. Roca et al., "Reed-Solomon Forward Error Correction (FEC) Schemes," RFC 5510, 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5510

[33] ——, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME," RFC 6865, 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6865