

---

This is an electronic reprint of the original article.  
This reprint may differ from the original in pagination and typographic detail.

Alhilal, Ahmad; Wu, Ze; Kämäräinen, Teemu; Braud, Tristan; Siekkinen, Matti

## Congestion Control for VR Cloud Gaming : Integration and Comparison in Real VR Gaming Environment

*Published in:*

MM '25: Proceedings of the 33rd ACM International Conference on Multimedia

*DOI:*

[10.1145/3746027.3755439](https://doi.org/10.1145/3746027.3755439)

Published: 27/10/2025

*Document Version*

Publisher's PDF, also known as Version of record

*Published under the following license:*

CC BY

*Please cite the original version:*

Alhilal, A., Wu, Z., Kämäräinen, T., Braud, T., & Siekkinen, M. (2025). Congestion Control for VR Cloud Gaming : Integration and Comparison in Real VR Gaming Environment. In *MM '25: Proceedings of the 33rd ACM International Conference on Multimedia* (pp. 12074-12082). ACM. <https://doi.org/10.1145/3746027.3755439>

---

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.



# Congestion Control for VR Cloud Gaming: Integration and Comparison in Real VR Gaming Environment

Ahmad Alhilal  
Aalto University  
Espoo, Finland

Ze Wu  
Hong Kong University of Science and  
Technology  
Hong Kong, Hong Kong

Teemu Kämäräinen  
University of Helsinki  
Helsinki, Finland

Tristan Braud  
Hong Kong University of Science and  
Technology  
Hong Kong, Hong Kong

Matti Siekkinen  
Aalto University  
Espoo, Finland

## Abstract

Virtual reality (VR) cloud gaming is increasingly developing in the gaming industry. Yet, the performance of the congestion control algorithms on top of which these systems build remains under-explored. In this study, we implement two industry-standard network congestion control algorithms, Google Congestion Control (GCC) and Network-Assisted Dynamic Adaptation (NADA), according to their Requests for Comments (RFCs), and integrate them into an open-source VR gaming system (ALVR). Including ALVR's congestion control (ALVR-ABR), we conduct extensive experiments on real-world networks to evaluate each algorithm's frame latency, target-to-receiving bitrate gap, dropped frames, image quality, and fairness among heterogeneous competing flows. GCC decreases frame latency by 35% compared to NADA and by 42% compared to ALVR. NADA and ALVR-ABR present significant gaps between the selected and received bitrate, causing substantial congestion-induced frame drops, while GCC has a minimal gap, resulting in minor frame drops, suggesting its suitability for game-player interaction. GCC exhibits a 2.7% and 5% decrease in image quality compared to NADA and ALVR-ABR, respectively, indicating slight immersion degradation. However, only NADA ensures a fair bandwidth share against loss-based flows due to its bitrate response to loss-induced congestion signals and lower sensitivity to delay gradients compared to GCC.

## CCS Concepts

• **Networks** → **Network performance analysis**; *Application layer protocols*; *Network mobility*; *Cloud computing*; • **Computing methodologies** → **Image compression**; **Virtual reality**.

## Keywords

Congestion Control, VR Cloud Gaming, Adaptive Video Encoding, Real-time Streaming.

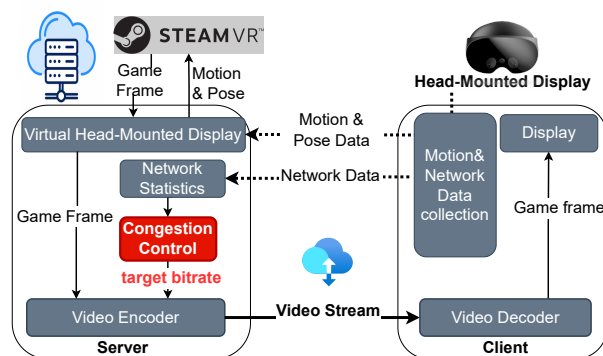


Figure 1: Congestion control in VR cloud gaming. Computing the target bitrate for encoding game frames based on the network performance on the client.

## ACM Reference Format:

Ahmad Alhilal, Ze Wu, Teemu Kämäräinen, Tristan Braud, and Matti Siekkinen. 2025. Congestion Control for VR Cloud Gaming: Integration and Comparison in Real VR Gaming Environment. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25)*, October 27–31, 2025, Dublin, Ireland. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3746027.3755439>

## 1 Introduction

Virtual reality (VR) cloud gaming enables users to play VR games on lightweight devices. It offloads the heavy computation (rendering) to the cloud [10, 19], eliminating the need for expensive hardware. However, streaming VR game content from the cloud to end-user devices requires stable, high-bandwidth network connections to maintain acceptable latency and playability. While video compression can reduce the bitrate, it is insufficient to ensure optimal performance under unstable or low-capacity network conditions. As shown in Figure 1, this work addresses the network congestion challenges for delivering a seamless VR gaming experience.

Internet data flows require congestion control to ensure fair bandwidth utilization and prevent network congestion collapse [18, 20, 26]. The IETF RMCAT Working Group has standardized two congestion control standards for real-time media: Google Congestion



This work is licensed under a Creative Commons Attribution 4.0 International License. *MM '25, Dublin, Ireland*

© 2025 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-2035-2/2025/10  
<https://doi.org/10.1145/3746027.3755439>

Control (GCC) [15] and Cisco’s Network-Assisted Dynamic Adaptation (NADA) Scheme [26]. Both algorithms (NADA and GCC) use one-way delay to detect congestion, with GCC calculating inter-frame delay differences and NADA measuring absolute frame delays. Numerous studies [16, 22, 24] have proposed congestion control algorithms and evaluated their performance in video streaming applications, including video-on-demand and live streaming. Some works focus on congestion control for cloud gaming [9, 23]. Additionally, several studies have investigated the performance of network congestion control algorithms [12, 13, 25]. These studies, however, mostly evaluate performance in simulated environments or in conventional video streaming. There has yet to be a thorough assessment of performance in real-world immersive applications, particularly VR cloud gaming. Unlike traditional cloud gaming (mobile or PC-based) [9, 23], VR cloud gaming poses greater challenges due to the close proximity of displays to players’ eyes, requiring higher resolutions (UHD) and extremely low latency to enhance immersion and minimize VR motion sickness.

In this work, we focus on the following research questions: (i) How can we incorporate standard congestion control algorithms for optimized game streaming and fair evaluation in VR cloud gaming? (ii) Which industry-standard network congestion control algorithm performs better in an immersive gaming environment? (iii) How do they perform in the presence of competing heterogeneous flows?

To answer the RQs, we implement two industry-standard network congestion control algorithms, GCC and NADA, based on their Requests for Comments (RFCs). We then integrate them into an open-source VR gaming platform called Air Light VR (ALVR). ALVR offers an adaptive bitrate (ABR) mode, which serves as ALVR’s congestion control algorithm (ALVR-ABR). To ensure game independence, we utilize ALVR base code [2] to integrate a virtual head-mounted display that interoperates with SteamVR [1], a VR game engine. This allows users to play any VR game with the specified congestion control algorithm. For reproducibility, we release the algorithms’ source code <sup>1</sup> for the research community to build upon. We then conduct extensive experiments to evaluate the algorithms’ performance in terms of frame latency and loss for high game-player interaction, and bitrate and image quality for immersion. Our contribution is summarized as follows:

- **Implement** two industry-standard congestion control algorithms (GCC and NADA) according to their RFCs and integrate them into an open-source VR cloud gaming platform.
- **Evaluate** the algorithms’ performance by examining latency, goodput, and image quality through extensive testing on realistic network setups. GCC presents the lowest latency, target-to-receive bitrate gap, and congestion-induced frame drops, making it ideal for seamless player interaction. However, GCC presents a slight decrease in image quality that may affect immersion.
- **Assess** their fairness towards heterogeneous competing flows. Only NADA achieves a fair bandwidth share with loss-based flows.

## 2 Background and Related Work

VR game engines typically generate UHD (+4K) composite game frames of left and right eye frames. VR Cloud Gaming encounters challenges related to the variation in network performance

and meeting high-quality experiences (interaction and immersion). Video compression, thus, is insufficient to ensure optimal performance under these variations. Furthermore, VR gaming is a time-sensitive and immersive media application, making it a compelling use case for understanding the performance of standard congestion control algorithms.

Web Real-Time Communication (WebRTC) [4] enables real-time communication in web browsers and mobile applications. It supports low-latency video communication through Google Congestion Control (GCC), which prevents network congestion collapse. GCC adjusts the transmission bitrate based on one-way delay gradients and packet loss. Network Assisted Dynamic Adaptation (NADA), developed by Cisco Systems, ensures real-time communication for media transmission. NADA uses two distinct rate controllers: one determines the desired bitrate, and the other sets the actual sending bitrate with a rate-shaping buffer to accommodate any temporary mismatch between them.

Huang et al [16] leverages reinforcement learning with human feedback to optimize the quality of experience (QoE) for adaptive video streaming. Reparo [22] enhances live video streaming by strategically discarding video frames. Reparo attempts to minimize the impact on video quality while maximizing bandwidth savings to improve users’ QoE on low-capacity networks. Karma [24] is an ABR algorithm that utilizes causal sequence modeling to improve generalization by comprehending the interrelated causality among past observations, returns, and actions and timely refining action when deviation occurs. These works mainly concentrate on congestion control in traditional video streaming. Pudica [23] and Nebula [9] are end-to-end congestion control algorithms for cloud gaming. Pudica strives to achieve near-zero queuing delay and high link utilization while respecting cross-flow fairness. It utilizes the paced frame to probe the bandwidth utilization ratio. Nebula adapts the video rate to the available bandwidth and applies frame-level redundancy to avoid visual distortions. While related works, including Pudica and Nebula, investigate methods for optimizing QoE and congestion avoidance, they do not assess performance in immersive environments (VR gaming settings). In contrast to traditional cloud gaming (mobile or PC-based), VR cloud gaming presents greater challenges due to the proximity of displays to eyes, requiring ultra-high resolutions and extremely low latency to improve immersion and minimize VR motion sickness.

Several studies have extensively examined the performance of congestion control algorithms for real-time media. Carlucci et al. [12, 13] analyze GCC’s performance in video conferencing applications by emulating a WAN scenario with multiple nodes and using a virtual webcam to send and receive a pre-recorded video. Carlucci et al. [14] compare the performance of NADA and GCC in a simulation environment. Zhang et al. [25] analyze the performance of three congestion control algorithms (Self-Clocked Rate Adaptation for Multimedia –SCReAM, NADA, and GCC) using the NS3 simulator. While the above studies extensively investigate congestion control algorithms for real-time media, they are limited to simulated environments. To our knowledge, no studies have examined the performance of these algorithms in the bandwidth-intensive, extremely time-sensitive, and immersive setting of VR cloud gaming.

<sup>1</sup>[https://github.com/ahmad-hl/GCC\\_NADA\\_ALVR](https://github.com/ahmad-hl/GCC_NADA_ALVR)

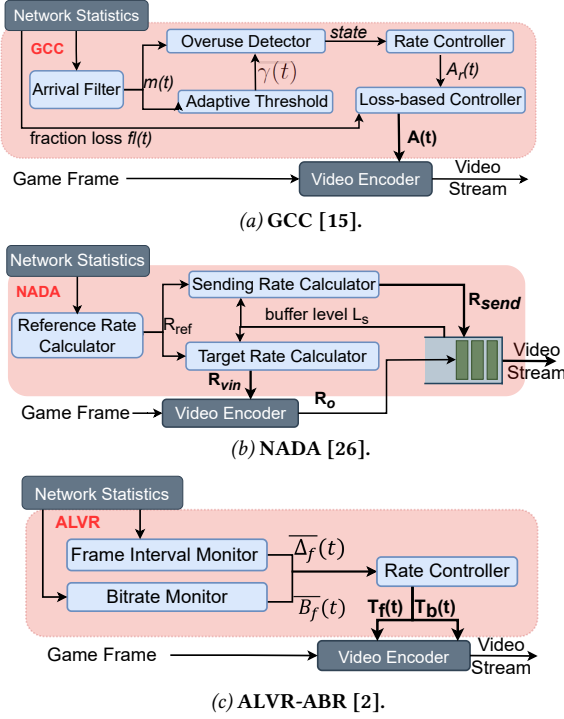


Figure 2: Congestion control mechanisms at the server.

For enhanced reproducibility and to contribute to the widespread adoption of VR cloud gaming, we implement two congestion control algorithms based on their RFCs and integrate them into the open-source VR cloud gaming platform (ALVR). This enables real-time monitoring of network data to allow the algorithms to estimate the desired bitrate and adjust video compression for game scenes. We then thoroughly evaluate their performance to determine the level of interaction and immersion they offer.

### 3 Congestion Control Algorithms

The Congestion Control module (shown in red in Figure 1) incorporates standard congestion control algorithms (GCC and NADA) into the VR cloud gaming system. The algorithm computes the target bitrate to dynamically adjust the video encoder's bitrate.

#### 3.1 Google Congestion Control (GCC)

GCC computes the target bitrate for the video encoder using network metrics reported by the client. As shown in Figure 2a, the computation employs four components: Arrival-time Filter, Overuse Detector, Remote Rate Controller, and Adaptive Threshold [12]. **Arrival Time Filter** estimates the queuing delay gradient  $m(t_i)$  of each video frame using the line slope formed by the sequence of data points  $(t_a(f_i), \Delta T_{delay}(f_i))$ , where  $t_a(f_i)$  is the arrival time of frame  $i$  and  $\Delta T_{delay}(f_i)$  is the inter-delay time. The delay gradient  $\Delta T_{delay}(f_i)$  is computed as the difference between the inter-arrival time and inter-departure time of frame  $i$  as follows:

$$\Delta T_{delay}(f_i) = [t_a(f_i) - t_a(f_{i-1})] - [t_d(f_i) - t_d(f_{i-1})] \quad (1)$$

The inter-arrival time is the time difference between the arrival of the current frame,  $t_a(f_i)$ , and the previous frame  $t_a(f_{i-1})$ . Likewise, the inter-departure time is the time difference between the sending of the current frame  $t_d(f_i)$ , and the previous frame  $t_d(f_{i-1})$ . We then compute the **adaptive threshold**  $\gamma(t_i)$  to account for the delay variations based on the network conditions.

**Over-use Detector** compares the delay gradient  $m(t_i)$  with the adaptive threshold  $\gamma(t_i)$ :

$$State = \begin{cases} Overuse & \text{if } m(t_i) > \gamma(t_i), \\ Normal & \text{if } -\gamma(t_i) \leq m(t_i) \leq \gamma(t_i), \\ Underuse & \text{if } m(t_i) < -\gamma(t_i). \end{cases} \quad (2)$$

**Remote rate controller** adjusts the target bitrate,  $A_r(t_i)$ , based on the state produced by the Over-use Detector, where  $\alpha$  is 0.85 and  $\eta$  is 1.05. The bitrate update is defined as follows:

$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \text{if } Incr, \\ \alpha R_c(t_i) & \text{if } Decr, \\ A_r(t_{i-1}) & \text{if } Hold. \end{cases} \quad (3)$$

**Loss-Based Controller:** Every time  $t_k$ , the Network Statistics unit receives the fraction of lost packets  $fl(t_k)$  which is used to compute the sending rate  $A_s(t_k)$  as follows:

$$A_s(t_k) = \begin{cases} A_s(t_{k-1})(1 - 0.5fl(t_k)) & \text{if } fl(t_k) > 0.1, \\ 1.05A_s(t_{k-1}) & \text{if } fl(t_k) < 0.02, \\ A_s(t_{k-1}) & \text{otherwise.} \end{cases} \quad (4)$$

#### 3.2 Network-Assisted Dynamic Adapt (NADA)

As illustrated in Figure 2b, the NADA sender (Server) calculates the target bitrate for the video encoder rate  $R_{vin}$  and sending rate  $R_{send}$  separately. The receiver (Client) aggregates per-packet drops and one-way delay measurements. The receiver periodically reports the congestion signal  $x_{curr}$  and the receiving rate  $R_{recv}$  to the Network Statistics unit in the sender.

**Measurement and Monitor Network at Receiver.** The receiver computes the one-way queuing delay  $d_{queue}$  as follows:

$$d_{queue}(f_i) = [t_a(f_i) - t_d(f_i)] \quad (5)$$

It then calculates the equivalent delay  $\tilde{d}$  as follows:

$$\tilde{d} = \begin{cases} d_{queue}, & \text{if } d_{queue} < Q_{TH} \\ Q_{TH} \cdot \exp\left(-\lambda \frac{d_{queue} - Q_{TH}}{Q_{TH}}\right), & \text{otherwise.} \end{cases} \quad (6)$$

where  $\lambda$  is a scaling parameter, and  $Q_{TH}$  is the delay threshold for invoking non-linear warping. This warping is inspired by the delay-adaptive congestion window backoff policy to "gradually nudge" the controller to operate based on loss-induced congestion signals when competing against loss-based flows.

The composite congestion signal is calculated as follows:

$$x_{curr} = \tilde{d} + D_{LOSS} \left( \frac{P_{LOSS}}{PLR_{REF}} \right)^2 \quad (7)$$

$D_{LOSS}$  is a reference delay penalty associated with packet loss estimate  $p_{loss}$  at a reference packet loss ratio of  $PLR_{REF}$ . In the absence of packet losses,  $x_{curr}$  reduces to the observed queuing delay  $d_{queue}$ , causing operation in a delay-based adaptation mode.

**Rate Adaptation at Sender.** The sender updates the reference rate  $R_{ref}$  as a function of the congestion signal  $x_{curr}$ . As illustrated in Figure 2b, the sender adjusts the target rate  $R_{vin}$  and sending

rate  $R_{send}$  based on  $R_{ref}$  and the length of rate shaping buffer  $L_s$ . It updates  $R_{ref}$ , following either an accelerated ramp-up or gradual update. In accelerated ramp-up mode,  $R_{ref}$  is updated as follows:

$$R_{ref} = \max(R_{ref}, (1 + \gamma)R_{recv}). \quad (8)$$

where  $\gamma$  is the rate increase multiplier.

In gradual update mode,  $R_{ref}$  is updated as follows:

$$R_{ref} = R_{ref} - \kappa \frac{\delta}{\tau} \cdot \frac{x_{offset}}{\tau} R_{ref} - \kappa \cdot \eta \frac{x_{curr} - x_{prev}}{\tau} R_{ref} \quad (9)$$

The rate change is affected by the offset of the aggregate congestion signal from its value at equilibrium  $x_{offset}$  and its change  $x_{diff}$ . The  $x_{offset}$  value is calculated based on the maximum flow rate  $R_{max}$ , its weight of priority  $PRIO$ , and the reference congestion signal  $X_{REF}$ . The value of  $X_{REF}$  is selected to achieve  $R_{max}$  when the congestion signal is below  $PRIO * X_{REF}$ . The scaling parameter  $\eta$  (set at 2.0) is used for the gradual rate update.

**The rate-shaping buffer** absorbs the instantaneous mismatch between the encoder's output bitrate  $R_o$  and the regulated sending rate  $R_{send}$ , with a buffer size  $L_s$ . The encoder target bitrate  $R_{vin}$  and the sending bitrate  $R_{send}$  are updated based on  $R_{ref}$  and  $L_s$  as follows:

$$\begin{aligned} R_{vdiff} &= \min(0.05 \cdot R_{ref}, \beta_v \cdot 8 \cdot L_s \cdot FPS) \\ R_{sdiff} &= \min(0.05 \cdot R_{ref}, \beta_s \cdot 8 \cdot L_s \cdot FPS) \\ R_{vin} &= \max(R_{min}, R_{ref} - R_{vdiff}) \\ R_{send} &= \min(R_{max}, R_{ref} + R_{sdiff}) \end{aligned} \quad (10)$$

The scaling parameters  $\beta_v$  and  $\beta_s$  can be tuned to balance the competing goals of maintaining a small rate-shaping buffer and deviating the system from the reference rate point.

### 3.3 ALVR Congestion Control (ALVR-ABR)

Figure 2c illustrates the ALVR's adaptive bitrate (ALVR-ABR). **Rate Controller** updates the target bitrate  $T_b$  and framerate  $T_f$  according to the mean values of monitored frame interval time  $\overline{\Delta_f}$  and the sending bitrate  $\overline{B_f}$ . **Frame Interval and Bitrate Monitor** units maintain sliding windows of the recent 256 frames. These windows store the time duration  $\Delta_f$  between consecutive frames  $f_i$  and  $f_{i-1}$ , as well as the bitrate of the  $i$ -th frame  $B_f$  calculated as the frame size divided by its network latency.

The controller calculates the target bitrate  $T_b$  and framerate  $T_f$  as:

$$T_b(t) = \beta \cdot \overline{B_f}(t), \quad T_f(t) = 1/\overline{\Delta_f}(t) \quad (11)$$

where  $\beta$  is the saturation multiplier (value 0.95), the desired percentage of available bandwidth to allocate for the video stream.

Table 1 outlines the adjustments to target bitrate for each congestion control algorithm. GCC and NADA consider one-way queuing delay gradients to compute the target bitrate. GCC computes the queuing delay as the time difference between consecutive frames' inter-arrival and inter-departure (see Equation 1), while NADA calculates the queuing delay as the difference between the frame's arrival and departure timestamps (see Equation 5). Meanwhile, ALVR adaptive bitrate mode (ABR) relies on network latency to compute the frame bitrate and derive the target bitrate.

**Table 1: Target Bitrate by GCC, NADA, and ALVR-ABR**

ABR Algorithm	Target Bitrate (Video Encoder Parameter)
GCC	$A_r(t_i)$ : Updated according to consecutive frames' queuing delay and bandwidth usage (Equation 3) $A_s(t_k)$ : updated based on packet loss (Equation 4)
NADA	$R_{vin}$ : Updated according to congestion signal $x_{curr}$ , computed on frame's queuing delay and packet loss (Equation 10)
ALVR ABR	$T_b(t)$ : Updated according to frame's bitrate $B_f$ , computed as frame size divided by network latency (Equation 11)

## 4 Implementation

### 4.1 System Architecture

Figure 1 outlines the end-to-end architecture for the VR cloud gaming system. We integrate GCC and NADA for adaptive game streaming and evaluate them against ALVR-ABR. This integration not only enables fair performance evaluation across benchmarks but also ensures game-agnostic VR cloud gaming. By interoperating with SteamVR, a VR game engine, the system allows users to install and play any VR game. The architecture consists of a server application (Windows) on the cloud and a client application (Android) on the VR device. The server encompasses four primary components: a virtual head-mounted device (VHMD), rendering, video encoding, and the congestion control component. The **VHMD** module replicates the physical HMD by receiving the physical motion data from the client and replaying them to SteamVR, the VR game engine. SteamVR then renders the corresponding game frame and returns the game frame to VHMD which composites the left eye and right eye frames into a single frame (game frame). The **Video Encoder** encodes it based on the target bitrate dictated by the **Congestion Control** algorithm. The encoded frame is transmitted to the client, which receives the video packets, decodes them to recover the frame, and plays back the recovered frame. Simultaneously, the client monitors the player's physical pose and motion (hand controller input and head rotation) and the network performance using **Motion & Network Data Collection** Unit. Clients constantly report this data to the server. Motion data feeds the VHMD module for rendering the corresponding game scene, while network data feeds the Network Statistics module, which the Congestion Control module uses to adjust the target bitrate.

### 4.2 Network Performance & Video Coding

The system encompasses server and client modules (see Figure 1). Most components operate on the server to lower the client's computational load. To achieve game-agnostic rendering and streaming, we implement the system on the ALVR base code [2]. The server records three key metrics: video receiving bitrate  $R_c(t)$  calculated as the sum of received packet sizes per 500ms interval, frame arrival time  $t_a(f_i)$ , and motion-to-photon latency, measuring the end-to-end delay from VR motion capture to frame display.

We use the ALVR code base to encode and decode the game frames. ALVR client utilizes the Android internal decoder with the corresponding encoding configuration received from the server. ALVR uses NVIDIA Encoder (NVENC), a hardware video encoder for Nvidia GPUs [3], with H.264 set as the default video codec.

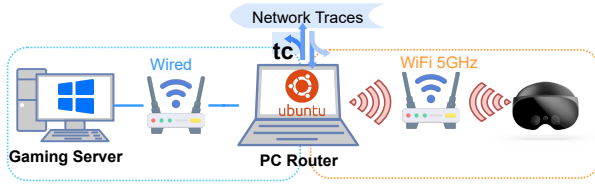


Figure 3: Cloud gaming emulator with a PC router incorporating mobile 5G network traces using traffic control (tc).

GCC, NADA, and ALVR-ABR produce target bitrates as primary parameters for encoding the game frames.

## 5 Evaluation

We thoroughly evaluate the algorithms’ performance in two use cases, a stable network (inspired by home WiFi) and 5G mobile network (driven by real-world deployment).

### 5.1 Experimental Setup

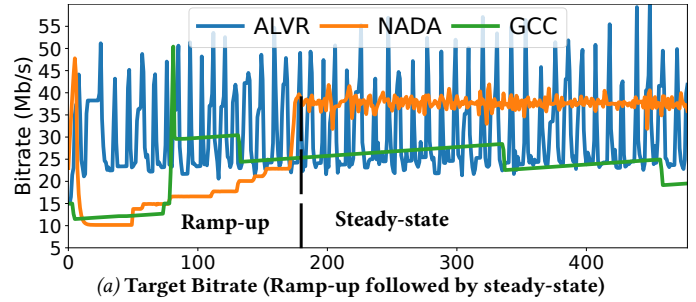
**Testbed.** As illustrated in Figure 3, we first design a physical testbed to emulate cloud gaming over real-life networks. We configure a Linux Laptop with 5GHz WiFi with a measured minimum speed of 128 Mb/s and Ethernet interfaces (Ubuntu 24.04 LTS, 11th Gen Intel(R) Core (TM) i5-1145G7x8) to function as a router between wired and wireless networks. This router relays traffic data between Meta Quest Pro (Android 12, Octa-core Kryo 585, Adreno 650) over a wireless network (5GHz WiFi) and a Windows PC (Windows 11 Pro, Intel(R) Core(TM) i5-1400 (16 CPUs) 2.5 GHz, NVIDIA GeForce RTX 4070 SUPER) over Ethernet to serve as a cloud server. We run a bash script on the PC router to emulate mobile network connectivity for the Meta Quest Pro. The script uses the Token Bucket Filter (TBF) in traffic control (tc) [8] to vary network throughput based on 5G mobile network traces [21] with link delay 5 ms [11]. TBF regulates the packet sending rate in the PC router by dropping packets to maintain the flow within the limits set by the traces.

**Evaluation Setup.** We play a first-person shooting game (Longbow) [7] in The Lab VR games on Steam [6]. We set the eye frame resolution to 2144x2336 (2K), resulting in a composite frame of 4288x2336 (over 4K). We configure ALVR to operate in adaptive bitrate (ABR) mode to compare it against GCC and NADA. We set the refresh rate to 72 Hz, resulting in a frame rate of 72 frame/sec. Notably, the ALVR open-source client is configured to request an intra-frame whenever a video frame is lost to prevent image distortion. Therefore, the client discards subsequent inter-frames until the intra-frame is received. This results in frame drops when packets are lost because of network congestion.

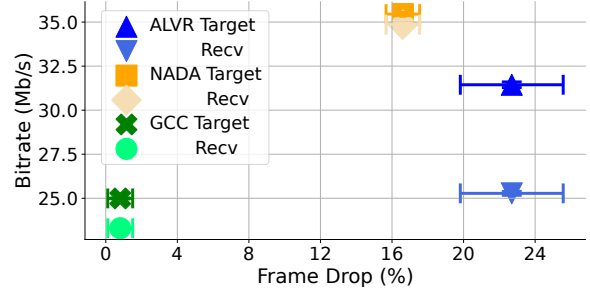
### 5.2 Stable Bandwidth

Ideally, VR game players use home WiFi, which tends to be stable. We allocate a fixed bandwidth to understand this behavior better and quantify the performance of each protocol. We set the bandwidth between the client and the server to 35 Mbps.

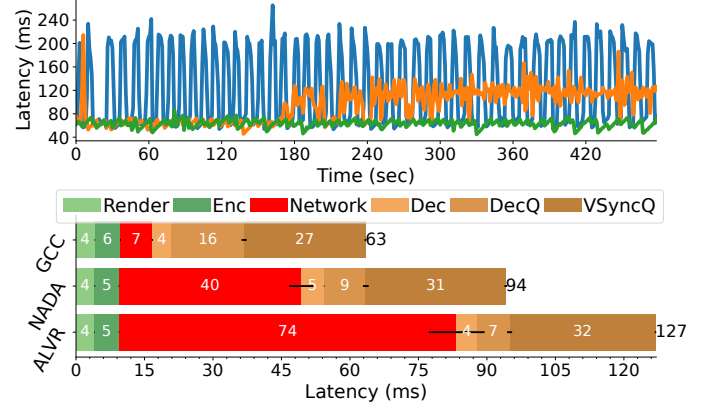
**Bitrate.** Figure 4a illustrates the target bitrate set by the congestion control algorithms examined. We note that delay gradient-based congestion control algorithms (NADA and GCC) operate in



(a) Target Bitrate (Ramp-up followed by steady-state)



(b) Target/Receive bitrate to frame drop with error bars (CI 95%)



(c) Frame latency (ms) with error bars (CI 95%)

Figure 4: Bitrate and latency of protocols over a stable bandwidth of 35 Mb/s, reported for three independent runs, an 8-minute gameplay session for each protocol.

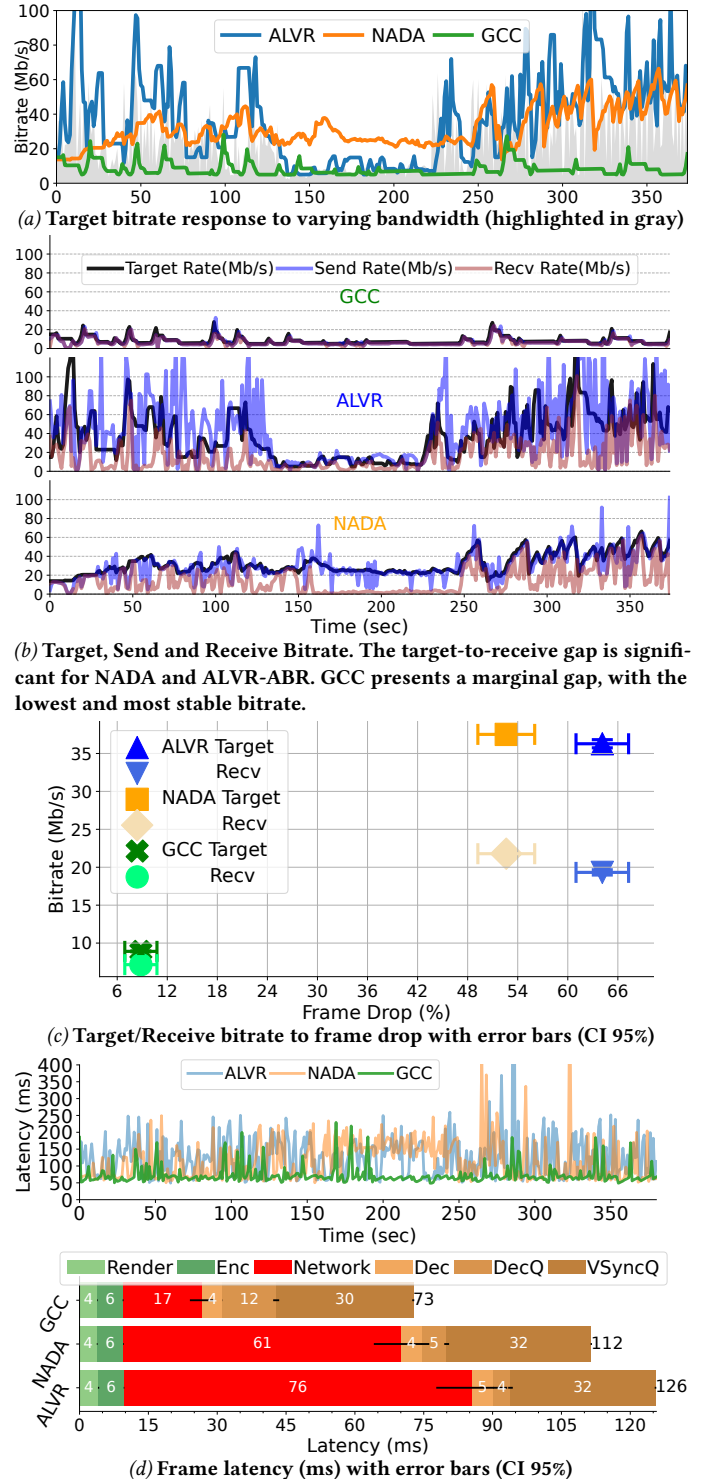
two distinct phases: the Ramp-up phase and the Steady-state phase. During the Ramp-up phase, they progressively adjust the bitrate to align with the available bandwidth (35 Mb/s), with GCC presenting a short spike exceeding the bandwidth at 95 seconds. They then maintain a steady-state bitrate, where NADA exceeds the bandwidth capacity and GCC underutilizes it. In contrast, ALVR-ABR displays a highly oscillatory pattern in bitrate changes, frequently swinging between overutilizing the available bandwidth and significantly underutilizing it. NADA overutilizes the available bandwidth during the steady-state phase, showing constant oscillation with minor fluctuations. Meanwhile, GCC tends to underutilize the available bandwidth but demonstrates the highest bitrate stability.

To understand the cascading effects of bitrate selection, we examine the difference between the receiving and the target bitrates,

along with frame loss resulting from network congestion. Figure 4b shows a scatter plot of the target and receiving bitrates (y-axis) against the percentage of frame drops (x-axis). We observe a significant gap between the target and receiving bitrate in ALVR-ABR, and a marginal gap in GCC and NADA. ALVR-ABR has an average target bitrate of  $31.4 \pm 0.16$  Mb/s and a receiving bitrate of  $25.3 \pm 0.1$  Mb/s, leading to a congestion-induced frame drop of  $22.7 \pm 3\%$ . NADA average target bitrate is  $35.5 \pm 0.1$  Mb/s, with a slightly lower receiving bitrate of  $34.8 \pm 0.1$  Mb/s, resulting in less congestion and a frame drop of  $16.6 \pm 1\%$ . GCC has the lowest average target bitrate at  $25 \pm 0.03$  Mb/s and a receiving bitrate of  $23.3 \pm 0.03$  Mb/s, causing a frame drop of  $0.8 \pm 0.7\%$ . Our statistical tests using One-way Analysis of Variance (ANOVA) followed by Tukey’s Honestly Significant Difference (HSD) reveal a significant difference ( $p < 0.001$ ) among the congestion control protocols.

**Frame Latency.** Figure 4c depicts the frame latency of the algorithms, which includes rendering (Render) time, video encoding (Enc) delay at the server, network delay, and the delays related to video decoding (Dec), queuing (DecQ), and Vertical Sync Queuing (VSyncQ) at the client. ALVR introduces controlled delays (VSyncQ) to synchronize frame rendering with the display refresh rate. Delay gradient-based algorithms (NADA and GCC) maintain low and stable latency during the Ramp-up stage. While GCC exhibits low latency in its Steady-state phase due to underutilization of bandwidth and stable bitrate, NADA faces significantly higher latency and fluctuating latency (mean 94 ms) due to aggressive bitrate increase aimed at dynamically probing network bandwidth. ALVR-ABR shows the highest latency and pronounced oscillation (mean 127 ms) due to its highly oscillatory pattern in bitrate changes. GCC presents the lowest latency of an average of 63 ms, with the lowest network latency of 7 ms, while NADA presents a higher latency of an average of 94 ms, with a significantly higher network latency of 40 ms. ALVR-ABR exhibits the highest latency of an average of 127 ms, with the highest network latency of 74 ms.

**One-way Forward Delay (OWD).** To understand the cascading effects of each algorithm’s bitrate selection on the frame latency, we examine the one-way forward delay of frames for each algorithm. As summarized in Table 1, NADA and GCC determine bitrate primarily based on frames’ one-way forward delay (OWD), while ALVR-ABR employs network latency to estimate the bitrate. We measure each protocol’s OWD as the difference between each delivered game frame’s arrival and departure timestamps. We note that GCC experiences a slight OWD spike during the Ramp-up phase, after which it stabilizes in the Steady-state phase with an OWD of  $0.7 \pm 1.9$  ms. NADA also experiences a small spike in OWD during the Ramp-up phase, followed by regular spikes in the Steady-state phase due to its aggressive bandwidth-probing strategy that leads to frequent queue buildups and a higher OWD of an average of  $3.75 \pm 9$  ms. In contrast, ALVR-ABR demonstrates the highest variability in OWD ( $5.3 \pm 22.5$  ms) in both the Ramp-up and Steady-state phases, attributed to its more aggressive bandwidth probing that leads to frequent congestion. Our statistical analysis using ANOVA and HSD shows a significant difference ( $p < 0.0001$ ), indicating notable OWD variations among the congestion control algorithms.



**Figure 5: Bitrate, latency and frame drop over 5G mobile network, reported for three independent runs, a 6-minutes gameplay session for each protocol. GCC exhibits the lowest bitrate, latency, target-to-receive bitrate gap, and minimal frame loss.**

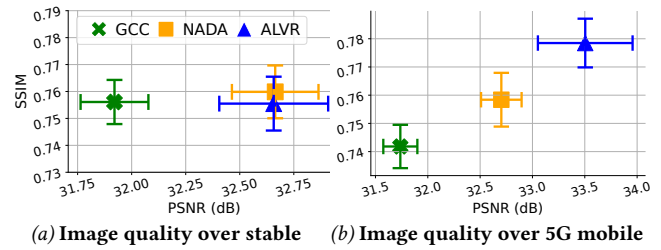
### 5.3 5G Network Bandwidth

**Bitrate Dynamics.** To assess adaptability, we utilize our testbed (section 5.1), which dynamically adjusts bandwidth according to 5G mobile network traces. We visualize the target bitrate of each congestion control protocol in Figure 5a. ALVR-ABR frequently overshoots in its attempts to utilize the available bandwidth, which is due to its low sensitivity to queuing delay and delayed response to congestion. NADA maintains a relatively lower target bitrate and experiences fewer overshooting events during high bandwidth variation. Interestingly, NADA shows more overshooting incidents when bandwidth variation is low. We attribute NADA’s failure to adapt to low bandwidth (140 sec-220 sec) to its congestion window backoff policy (see Equation 6 in Section 3.2), resulting in aggressive sending and continued pressure on the network. GCC, on the other hand, has the fewest overshooting events, thanks to its delay-based controller that considers the gradients of queuing delay. The gradients are calculated as the time difference between the frame’s inter-arrival and inter-departure, which can also lead to bandwidth underutilization.

Figure 5b illustrates the target, sending, and receiving bitrate over the gameplay time. ALVR-ABR shows significant deviations in its actual sending bitrate from the target, with instances of both higher and lower bitrates, indicating frequent congestion. Its receiving bitrate is usually lower than the target over time. NADA’s sending bitrate closely follows the target with minor deviations, suggesting less congestion, while its receiving bitrate also remains below the target. In contrast, GCC has the smallest gaps between sending, receiving, and target bitrates, indicating the least congestion.

We further examine the network congestion for each algorithm by analyzing the difference between the receiving bitrate and the target bitrate, and the resulting frame drop. Figure 5c shows a scatter plot of the target and receiving bitrates (y-axis) against the percentage of frame drops (x-axis) resulting from network congestion. We observe a significant gap between the target and receiving bitrate in NADA and ALVR-ABR, while the gap is marginal in GCC. ALVR-ABR has an average target bitrate of  $36.3 \pm 0.5$  Mb/s and a receiving bitrate of  $19.3 \pm 0.3$  Mb/s, leading to a significant congestion-induced frame drop of  $64 \pm 3.15\%$ . NADA’s average target bitrate is  $37.5 \pm 0.2$  Mb/s, with a significantly lower receiving bitrate of  $21.8 \pm 0.25$  Mb/s, resulting in less congestion and a frame drop of  $52.6 \pm 3.4\%$ . GCC has the lowest average target bitrate at  $8.9 \pm 0.05$  Mb/s and a receiving bitrate of  $7.15 \pm 0.05$  Mb/s, causing a frame drop of  $8.8 \pm 1.9\%$ . Notably, the video receiver at the client requests an intra-frame whenever a video frame is lost to prevent image distortion, discarding subsequent inter-frames until the intra-frame is received, which increases the frame drop ratio. Our ANOVA and HSD tests reveal a significant statistical difference ( $p < 0.0001$ ) among the algorithms.

**Latency.** Figure 5d illustrates frame latency of the protocols, which includes rendering (Render) time, video encoding (Enc) delay at the server, network delay, and the delays related to video decoding (Dec), queuing (DecQ), and VSync (VSyncQ) at the client. ALVR-ABR exhibits high end-to-end latency (mean 126 ms) with significant variations due to its low sensitivity to queuing delay and delayed response to congestion. Congestion resulting from NADA’s regular overshooting when bandwidth variation is low



**Figure 6: Target image quality (SSIM and PSNR) for (a) stable network and (b) 5G mobile network. With a slight difference, GCC shows the lowest quality, with slight differences.**

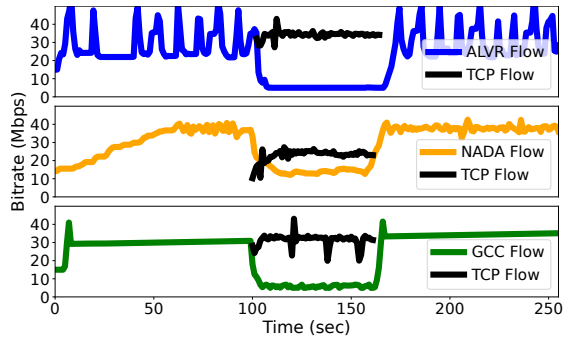
results in prolonged latency spikes, with an average of 112 ms. In contrast, GCC shows the lowest latency (mean 73 ms) with minimal spikes, thanks to its effective computation of queuing delay as the difference between the frame’s inter-arrival and inter-departure times. This is reflected in GCC’s lowest network latency (mean 17 ms), while NADA has a notably higher latency (mean 61 ms). ALVR-ABR displays the highest network latency (mean 76 ms). Notably, GCC’s lower bitrate results in more delivered frames awaiting decoding, which leads to longer queuing delays in the buffer and, consequently, a higher DecQ compared to benchmarks.

**One-way Forward Delay (OWD).** We measure each protocol’s OWD for each delivered game frame. We note that GCC presents the lowest OWD of  $1.4 \pm 7.6$  ms and exhibits minimal variation in OWD over time. NADA follows with an average delay of  $6.5 \pm 27$  ms, due to NADA’s higher level of overshooting. ALVR-ABR demonstrates the highest delay between frame transmission and delivery, due to ALVR-ABR’s significant overshooting with an average of  $8.5 \pm 29.7$  ms. Our statistical tests using One-Way ANOVA and Tukey’s HSD reveal a significant difference ( $p < 0.002$ ), indicating that OWD values vary significantly across GCC, NADA, and ALVR-ABR.

**Target Image Quality.** We use Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) to assess the image quality of game frames encoded at the algorithm’s target bitrate. During gameplay, we capture both the rendered and encoded frames on the server. We then reconstruct the frame from the encoded version and compare it to the recorded rendered frame. Figure 6 illustrates the SSIM (y-axis) and PSNR (x-axis) for both stable WiFi network (Figure 6a) and mobile network setups (Figure 6b). GCC presents the lowest image quality in both metrics, over both network setups. ALVR-ABR achieves the highest SSIM and PSNR over mobile networks, while it is slightly lower than that of NADA under a stable bandwidth setup. This is attributed to the target bitrate allocated for each algorithm, which is the highest for ALVR-ABR, followed by NADA, while GCC is the lowest over the 5G mobile network. NADA tends to assign the highest bitrate under stable networks, making NADA exhibit a slightly higher SSIM and PSNR, compared to ALVR-ABR. Our ANOVA and HSD tests reveal a significant difference ( $p < 0.04$ ) among the algorithms.

### 5.4 Protocol Fairness

Protocol fairness refers to the capacity to share the available bandwidth fairly when competing for the same network resources. This



**Figure 7: Bitrate in the presence of competing heterogeneous flow; only NADA achieves a relatively fair bandwidth share.**

principle prevents any protocol from monopolizing the link to the detriment of others. In this experiment, we start two data flows—one using a congestion control protocol and the other using TCP—at different times on a bottleneck link. We utilize the IPerf3 tool [5] to generate the TCP flow. We use an external PC to serve as an IPerf3 client, while the gaming server serves as the IPerf3 server. We start the TCP flow 100 seconds after the protocol flow.

Figure 7 illustrates the target bitrate of each algorithm and the bitrate of competing TCP flow under a link capacity of 35 Mbps. NADA receives a higher, yet lower than half, bandwidth share. GCC is highly sensitive to latency and packet loss, prioritizing low latency at the cost of reduced bitrate [17]. As such, TCP often takes a larger share of bandwidth. ALVR-ABR uses historical per-frame bitrate observations to calculate its target bitrate. Congestion caused by TCP flows leads to delays in frame delivery, reducing the frame bitrates  $B_f$  and ultimately reducing ALVR-ABR’s share. The large bandwidth share of TCP stems from its reliance on a loss-based congestion control algorithm. Loss-based algorithms increase transmission rates until they detect a congestion signal (packets lost), increasing the RTT of other flows and triggering the delay-based flow to back off. Flows of delay-based algorithms experience a huge decrease in throughput if they share the bottleneck with loss-based algorithms. This occurs because they detect congestion sooner, right when the queues begin to fill. Since ALVR-ABR is not loss-based, and GCC is delay-based, their bandwidth share drops significantly. Only NADA maintains a fairly equitable bandwidth share because it is less sensitive to delay gradients and uses non-linear warping (Equation 6) to adjust the controller to respond to loss-induced congestion signals.

## 6 Findings and Discussion

We examined industry-standard network congestion control algorithms for immersive media streaming in a real-life VR gaming environment. We implemented two algorithms based on their RFCs: Network-Assisted Dynamic Adaptation (NADA) from Cisco Systems and Google Congestion Control (GCC) from Google. We then incorporated them into an open-source VR gaming platform (ALVR). We evaluated their performance against ALVR’s congestion control (ALVR-ABR) over real-world networks.

ALVR-ABR ignores queuing delay gradients in bitrate adaptation, causing sluggish congestion responses that trigger frequent

bitrate fluctuations. GCC demonstrates the most stable performance through its delay-sensitive control mechanism, while NADA shows marginally more overshooting incidents. The difference in bitrate selection between NADA and GCC stems from their one-way delay measurements: NADA measures the time from a frame’s departure to arrival, while GCC measures the time difference between the frame’s inter-arrival and inter-departure. On the one hand, GCC’s delay-sensitive bitrate control results in the lowest and most stable frame latency, followed by NADA, whereas ALVR-ABR exhibits the highest latency and significant fluctuations. This positions GCC as the best option for optimal game-player interaction in VR cloud gaming. On the other hand, NADA presents the highest image quality, and ALVR-ABR follows over mobile networks, whereas GCC presents the lowest image quality. GCC thus may slightly reduce immersion, as higher image quality is vital for player engagement. However, both NADA and ALVR-ABR suffer significant frame drops due to frequent overshooting, resulting in high network congestion. This is evident in the substantial gap between the target and received bitrate for NADA and ALVR-ABR, whereas GCC shows a marginal gap over the mobile network. In terms of fairness, when competing with loss-based (TCP) flow, only NADA maintains a fair bandwidth share because it is less sensitive to delay gradients and its aggressive bitrate allocation in response to loss-induced congestion signals.

Building on GCC, future congestion controllers should incorporate delay gradients while minimizing delay sensitivity. This approach would help utilize available bandwidth more efficiently, resulting in higher image quality. Inspired by NADA, the controller should gradually adjust to operate based on loss-induced congestion signals when competing with loss-based flows. To support this initiative and enhance reproducibility, we have made the source code of the algorithms publicly available<sup>2</sup> to the research community.

## 7 Conclusion

Although VR cloud gaming is increasingly developing, the performance of the underlying congestion control algorithms remains under-explored. We implemented two industry-standard network congestion control algorithms—NADA from Cisco and GCC from Google—within an open-source VR cloud gaming platform (ALVR) and conducted extensive experiments over real-life WiFi and mobile networks. Our findings revealed that GCC offers the lowest and most stable bitrate, resulting in minimal latency and frame drop, which makes it well-suited for player-game interactions in VR cloud gaming. However, it also has the lowest image quality, which could diminish the visual experience. Conversely, NADA presents the highest bitrate, leading to overshooting events and a higher latency and frame drop. NADA’s aggressive bitrate allocation results in the highest image quality, but significant frame drops hinder immersion. Although ALVR-ABR presents a slightly lower bitrate, it experiences the most overshooting, leading to higher latency and frame drops. This compromises immersion despite delivering better image quality for transmitted frames. Only NADA achieves an equitable bandwidth share when competing with loss-based flows, while ALVR-ABR and GCC receive minimal shares.

<sup>2</sup>[https://github.com/ahmad-hl/GCC\\_NADA\\_ALVR](https://github.com/ahmad-hl/GCC_NADA_ALVR)

## Acknowledgments

This research was partially supported by a grant from the Hong Kong Innovation and Technology Commission under the Innovation and Technology Fund (ITS/319/22FP) and a project from the HKUST-Bright Dream Robotics Joint Research Institute (FTRIS-23-015). This work was also partially supported by the Research Council of Finland (grant number 355577).

## References

- [1] 2018. SteamVR. <https://store.steampowered.com/app/250820/SteamVR/> Access date: March 25, 2025.
- [2] 2023. Air Light VR (ALVR). <https://github.com/alvr-org/ALVR>
- [3] 2023. NVENC Video Encoder API Programming Guide. <https://docs.nvidia.com/video-technologies/video-codec-sdk/12.1/nvenc-video-encoder-api-prog-guide/index.html> Access date: March 25, 2025.
- [4] 2023. Real-time Communication for the Web (WebRTC). <https://webrtc.org/>
- [5] 2025. iPerf3: The ultimate speed test tool for TCP, UDP and SCTP. <https://iperf.fr/>
- [6] 2025. The Lab: a compilation of Valve's room-scale VR experiments. [https://store.steampowered.com/app/450390/The\\_Lab/](https://store.steampowered.com/app/450390/The_Lab/) Access date: March 25, 2025.
- [7] 2025. The Lab: Longbow (VR Gameplay). <https://steamcommunity.com/sharedfiles/filedetails/?id=883935639> Access date: March 25, 2025.
- [8] 2025. Traffic Control(tc) - Token Bucket Filter (tbf). <https://linux.die.net/man/8/tc-tbf>
- [9] Ahmad Alhailal, Tristan Braud, Bo Han, and Pan Hui. 2022. Nebula: Reliable Low-Latency Video Transmission for Mobile Cloud Gaming. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) (*WWW '22*). Association for Computing Machinery, New York, NY, USA, 3407–3417. <https://doi.org/10.1145/3485447.3512276>
- [10] Ahmad Alhailal, Ze Wu, Yuk Hang Tsui, and Pan Hui. 2024. FovOptix: Human Vision-Compatible Video Encoding and Adaptive Streaming in VR Cloud Gaming. In *Proceedings of the 15th ACM Multimedia Systems Conference* (Bari, Italy) (*MMSys '24*). Association for Computing Machinery, New York, NY, USA, 67–77. <https://doi.org/10.1145/3625468.3647612>
- [11] Tristan BRAUD, Pengyuan ZHOU, Jussi KANGASHARJU, and Pan HUI. 2020. Multipath Computation Offloading for Mobile Augmented Reality. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–10. <https://doi.org/10.1109/PerCom45495.2020.9127360>
- [12] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio 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* (Klagenfurt, Austria) (*MMSys '16*). Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. <https://doi.org/10.1145/2910017.2910605>
- [13] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2017. Congestion Control for Web Real-Time Communication. *IEEE/ACM Transactions on Networking* 25, 5 (2017), 2629–2642. <https://doi.org/10.1109/TNET.2017.2703615>
- [14] Gaetano Carlucci, Luca De Cicco, Cesar Ilharco, and Saverio Mascolo. 2016. Congestion control for real-time communications: A comparison between NADA and GCC. In *2016 24th Mediterranean Conference on Control and Automation (MED)*. 575–580. <https://doi.org/10.1109/MED.2016.7535978>
- [15] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. [n.d.]. A Google Congestion Control Algorithm for Real-Time Communication draft-ietf-rmcat-gcc-02. <https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc> Access date: March 25, 2025.
- [16] Tianchi Huang, Rui-Xiao Zhang, Chenglei Wu, and Lifeng Sun. 2023. Optimizing Adaptive Video Streaming with Human Feedback. In *Proceedings of the 31st ACM International Conference on Multimedia* (Ottawa ON, Canada) (*MM '23*). Association for Computing Machinery, New York, NY, USA, 1707–1718. <https://doi.org/10.1145/3581783.3611771>
- [17] Hassan Iqbal, Ayesha Khalid, and Muhammad Shahzad. 2021. Dissecting Cloud Gaming Performance with DECAF. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 31 (dec 2021), 27 pages. <https://doi.org/10.1145/3491043>
- [18] Randell Jesup and Zaheduzzaman Sarker. [n.d.]. Congestion Control Requirements for Interactive Real-Time Media RFC 8836. <https://datatracker.ietf.org/doc/rfc8836/> Access date: March 28, 2025.
- [19] Teemu Kämäräinen, Matti Siekkinen, Jukka Erikäinen, and Antti Ylä-Jääski. 2018. CloudVR: Cloud Accelerated Interactive Mobile Virtual Reality. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) (*MM '18*). Association for Computing Machinery, New York, NY, USA, 1181–1189. <https://doi.org/10.1145/3240508.3240620>
- [20] Y.N. Prajapati and Pranob Adhikari. 2022. Efficient Allocation of Bandwidth & Congestion Avoidance during Data Flow. In *2022 International Conference on Fourth Industrial Revolution Based Technology and Practices (ICFIRTP)*. 120–122. <https://doi.org/10.1109/ICFIRTP56122.2022.10059436>
- [21] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. 2020. Beyond Throughput, the next Generation: A 5G Dataset with Channel and Context Metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference* (Istanbul, Turkey) (*MMSys '20*). Association for Computing Machinery, New York, NY, USA, 303–308. <https://doi.org/10.1145/3339825.3394938>
- [22] Fulin Wang, Qing Li, Wanxin Shi, Gareth Tyson, Yong Jiang, Lianbo Ma, Peng Zhang, Yulong Lan, and Zhicheng Li. 2023. Reparo: QoE-Aware Live Video Streaming in Low-Rate Networks by Intelligent Frame Recovery. In *Proceedings of the 31st ACM International Conference on Multimedia* (Ottawa ON, Canada) (*MM '23*). Association for Computing Machinery, New York, NY, USA, 9194–9204. <https://doi.org/10.1145/3581783.3613441>
- [23] Shibo Wang, Shusen Yang, Xiao Kong, Chenglei Wu, Longwei Jiang, Chenren Xu, Cong Zhao, Xuesong Yang, Jianjun Xiao, Xin Liu, Changxi Zheng, Jing Wang, and Honghao Liu. 2024. Pudica: Toward Near-Zero Queuing Delay in Congestion Control for Cloud Gaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 113–129. <https://www.usenix.org/conference/nsdi24/presentation/wang-shibo>
- [24] Bowei Xu, Hao Chen, and Zhan Ma. 2023. Karma: Adaptive Video Streaming via Causal Sequence Modeling. In *Proceedings of the 31st ACM International Conference on Multimedia* (Ottawa ON, Canada) (*MM '23*). Association for Computing Machinery, New York, NY, USA, 1527–1535. <https://doi.org/10.1145/3581783.3612177>
- [25] Songyang Zhang, Weimin Lei, Wei Zhang, and Yunchong Guan. 2019. Congestion control for RTP media: A comparison on simulated environment. In *International Conference on Simulation Tools and Techniques*. Springer, 43–52.
- [26] Xiaoqing Zhu, Rong Pan, Michael A. Ramalho, and Sergio Mena de la Cruz. [n.d.]. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media RFC 8698. <https://datatracker.ietf.org/doc/rfc8698/>