NICoLE: Are In-Network LLM-Based Agents Cost-Feasible for RTP Video Streaming?

Alireza Shirmarz, Fábio Luciano Verdi, Gyanesh Patra, Gergely Pongrácz
{ashirmarz, verdi}@ufscar.br, {gyanesh.patra, gergely.pongracz}@ericsson.com

***Abstract*—We present NICoLE (Network-Integrated Codec–LLM Engine), an in-network decision layer for real-time RTP video streaming that assists encoder adaptation without modifying end-to-end congestion control. NICoLE correlates RTP telemetry (frame size and packet pacing) with queue dynamics at the bottleneck to detect mismatch between encoded rate and available service capacity. Based on this, it selects discrete encoder profiles (resolution, frame rate, GoP) and applies short-lived ECN-based queue prioritization. The system is implemented using WebRTC (GCC) and GStreamer. Experiments under controlled bottleneck conditions show reduced queue buildup, lower stall ratio, and improved frame pacing stability at high frame rates (90–120 FPS), while gains remain limited at lower rates. To address reliability concerns, NICoLE constrains the LLM to structured inputs and a bounded action space, ensuring deterministic behavior and avoiding free-form reasoning. We show that inference latency (∼383 ms) remains below GCC reaction time (600–1500 ms), enabling timely in-network intervention. These results indicate that LLM-assisted in-network control is feasible and beneficial under high-density, delay-sensitive workloads, but not universally required.**

***Index Terms*—AI Agent, Real Time Video Streaming, L4S, QoE.**

I. Introduction

RTP-based real-time video streaming underpins interactive applications such as cloud gaming and XR [1]–[4]. These systems operate under tight latency and jitter budgets, where transient queue buildup inflates motion-to-photon delay, disrupts frame pacing, and degrades Quality of Experience (QoE), particularly at high frame rates (60–120 FPS) [5], [6].

Recent work shows that improving real-time video performance requires both faster congestion feedback loops and better encoder adaptation. Tight-loop congestion control reduces reaction latency [7], while encoder-side designs improve rate–distortion behavior under varying bandwidth [8]. However, both remain end-host driven and do not leverage in-network observations. As a result, congestion is still detected after queue buildup, creating a mismatch between real-time network conditions and encoder configuration.

During transient congestion, queue buildup distorts packet departure timing before end-to-end control converges, degrading frame pacing and QoE, especially in high-FPS workloads [5]. However, intervening within this adaptation window is challenging: due to encryption and processing constraints, network devices observe only RTP-level metadata and queue state, without access to codec structure (e.g., resolution, FPS, GoP). This creates a synchronization gap between congestion onset and encoder reconfiguration.

Challenge 1: Bitrate-only control is insufficient. WebRTC's Google Congestion Control (GCC) [9] continuously adjusts bitrate based on delayed receiver feedback, while encoder parameters (resolution, FPS, GoP) evolve on slower timescales. During transient congestion, bitrate reduction alone does not immediately align the stream with instantaneous service capacity, leading to suboptimal operation before encoder adaptation converges.

Challenge 2: Encoder reconfiguration incurs transient mismatch. Even when adaptation is triggered, encoder reconfiguration and signaling require non-negligible time. During this interval, queue buildup and pacing distortion persist. While AQM mechanisms such as CoDel [10] or ECN-enabled L4S DualPI2 [11], [12] reduce queue delay, they operate at flow granularity and do not support short-lived prioritization aligned with encoder adaptation windows.

To address these challenges, the network must infer encoder pacing and service capacity directly at the bottleneck. RTP headers expose frame boundaries, timestamps, and packet sizes, enabling extraction of inter-frame gap (IFG) and frame size (FS) without payload inspection [4]. Combined with queue telemetry, this enables early detection of rate mismatch.

By correlating encoded rate with observed service capacity, the network can trigger profile-level adaptation at congestion onset. During the adaptation interval, temporary ECT(1) marking can be applied to leverage L4S prioritization for short-timescale mitigation. Unlike persistent L4S marking, this approach is transient and bounded.

This multi-dimensional decision problem—mapping RTP telemetry and queue state to discrete adaptation actions—extends beyond simple threshold-based control. Rule-based logic requires careful tuning across heterogeneous encoder behaviors, while conventional ML models rely on fixed feature mappings. In contrast, compact LLMs can map structured telemetry inputs to a bounded action space without manual threshold tuning [13]–[15]. This design avoids free-form reasoning and ensures predictable behavior suitable for network control.

Building on this direction, we propose *NICoLE*, an in-network LLM-based agent that assists real-time RTP streaming through (i) profile-aware encoder adaptation and (ii) transient L4S steering, while preserving end-to-end congestion control semantics.

ISBN 978-3-903176-82-9

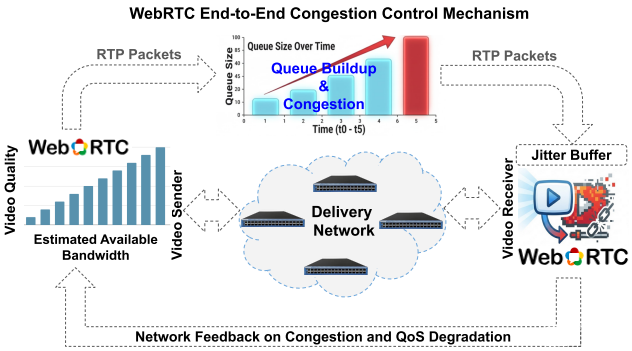


Fig. 1: WebRTC real-time streaming control loop.

NICoLE operates as a bounded decision layer that correlates RTP telemetry with queue dynamics. Upon transient congestion, it activates short-lived prioritization and selects a discrete encoder profile aligned with observed service capacity, complementing GCC's bitrate control.

This work presents a cost-aware feasibility study of network-assisted encoder configuration using compact LLM agents. We evaluate whether such in-network intelligence delivers measurable QoE gains while respecting inference latency and deployment constraints.

Our contributions are as follows:

- **In-Network LLM Control Framework:** A bounded decision architecture linking RTP telemetry and queue state to encoder-level adaptation without modifying end-to-end congestion control;
- **Transient L4S Steering:** Short-timescale prioritization during encoder adaptation;
- **Profile-Aware Encoder Assistance:** Network-informed selection of discrete encoder profiles (resolution, FPS, GoP);
- **Cost–Benefit Evaluation:** CPU-based evaluation of QoE, queue behavior, and inference feasibility.

This work does not claim universal superiority of LLM-based control, but demonstrates its feasibility and benefit under high-density, delay-sensitive workloads.

II. Background and System Overview

A. WebRTC Streaming and Control Limitations

Interactive applications such as cloud gaming and XR employ RTP over UDP to achieve low latency and fine-grained playout control [1], [2], [16]. Encoded video frames (e.g., H.264) are packetized into RTP packets using standardized formats [17]. Fig. 1 illustrates the WebRTC control loop. RTP packets traverse shared network devices where queue buildup may occur; the receiver jitter buffer absorbs delay variation; and RTCP feedback drives end-to-end bitrate adaptation.

WebRTC commonly employs Google Congestion Control (GCC) [9], which estimates available bandwidth from receiver feedback (e.g., TWCC delay trends [18]) and adjusts the sender bitrate. Transport control and codec behavior are decoupled: *GCC regulates bitrate*, while the *encoder determines how bitrate is distributed* across resolution, frame rate (FPS), GoP, and quantization.

Under congestion, bitrate reduction precedes encoder profile adaptation, which occurs on slower timescales. When the encoded rate exceeds instantaneous service capacity, queue buildup persists, distorting packet pacing—an effect amplified at high frame rates.

B. From Reactive Control to In-Network Insight

Congestion manifests within network queues, yet control remains reactive: the sender detects congestion only after delay inflation is observed at the receiver via RTCP feedback [9], [18], [19]. As a result, a mismatch arises between encoded rate and available service capacity during transient congestion.

Active Queue Management (AQM) mechanisms such as L4S reduce persistent queue delay using ECN-based dual-queue designs (e.g., DualPI2) [11], [12]. However, they typically operate at flow granularity and rely on persistent ECT(1) marking, without coordination with encoder adaptation.

At the same time, RTP headers expose frame-level structure through timestamps, marker bits, and packet sizes. These signals enable estimation of frame size (FS), inter-frame gap (IFG), and encoded rate directly in-network. By correlating these features with queue state, the network can detect service-rate mismatch early, before end-to-end control converges.

Design Insight. This observation enables a bounded in-network mechanism that: (i) infers encoder pacing and service capacity from RTP and queue telemetry, (ii) assists discrete encoder profile adaptation without modifying GCC semantics, and (iii) applies short-lived prioritization to suppress queue buildup during reconfiguration.

C. NICoLE System Overview

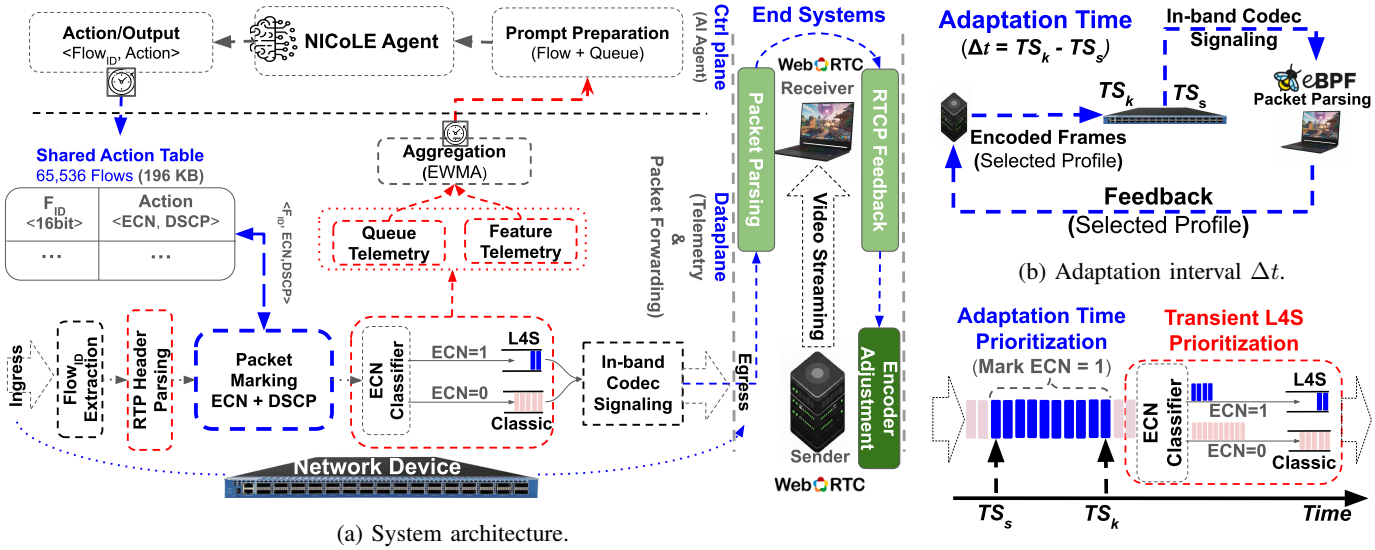
Fig. 2 presents NICoLE, an in-network decision layer that operates alongside WebRTC without modifying end-to-end congestion control. A control-plane agent periodically processes RTP and queue telemetry and updates a per-flow action table, while the dataplane enforces bounded actions without per-packet inference.

The system consists of a sender, a receiver, and an in-network NICoLE module. The sender performs video encoding and GCC-based bitrate adaptation, while the receiver maintains a jitter buffer and provides RTCP feedback. NICoLE augments this loop by extracting telemetry in the dataplane and invoking a control-plane decision module.

Telemetry includes RTP fields (sequence number, timestamp, marker bit, packet size), pacing signals (IFG_{server}, IFG_{switch}), frame size, and queue state. Pacing distortion is defined as $\Delta_{IFG} = \mathrm{IFG}_{switch} - \mathrm{IFG}_{server}$, and the encoded rate is approximated as

$$R_{enc} \approx \frac{FS}{\mathrm{IFG}_{server}} \approx FS \times FPS.$$

Congestion arises when $R_{enc} > \mu$, where μ is the estimated service rate.



(a) System architecture.

(b) Adaptation interval Δt.

(c) Transient L4S steering.

Fig. 2: NICoLE architecture. (a) Control/data-plane interaction. (b) Adaptation interval Δt. (c) Transient ECT(1)-based L4S steering during Δt.

Based on this observation, NICoLE executes bounded actions: (i) transient L4S steering via conditional ECT(1) marking during an adaptation interval Δt, (ii) selection of discrete encoder profiles aligned with service capacity, and (iii) signaling of adaptation hints via DSCP toward the receiver, which forwards them to the sender through a WebRTC DataChannel.

Unlike conventional reactive control, NICoLE detects mismatch in-network and applies early adaptation, stabilizing frame pacing during transient congestion.

To ensure reliability, the decision module is implemented as a bounded LLM: inputs are structured numeric telemetry, and outputs are restricted to a discrete action tuple (profile ID, L4S flag, DSCP). This design prevents free-form reasoning and ensures deterministic, cost-aware operation.

III. Control Algorithm

NICoLE performs periodic control based on aggregated RTP and queue telemetry, detecting service-rate mismatch and triggering bounded profile adaptation without modifying GCC.

At each control interval:

1) **Telemetry aggregation:** RTP frame size (FS), sender pacing (IFG_{server}), switch inter-arrival gap (IFG_{switch}), and queue occupancy are smoothed using EWMA ($\alpha = 0.1$);
2) **Congestion detection:** congestion is declared when encoded rate R_{enc} exceeds service rate μ, or pacing distortion ΔIFG exceeds threshold θ;
3) **LLM decision:** a bounded LLM selects an encoder profile aligned with μ and determines whether transient L4S activation is required;
4) **Enforcement:** the dataplane applies ECT(1) marking during adaptation window Δt, updates DSCP hints, and maintains actions until convergence or congestion clears.

Profile transitions are rate-limited to prevent oscillation. NICoLE adjusts encoder operating points within GCC's bitrate budget.

A. Congestion Detection

For each flow f:

$$\Delta_{IFG}^{(f)} = \mathrm{IFG}_{switch}^{(f)} - \mathrm{IFG}_{server}^{(f)} \tag{1}$$

Under uncongested conditions:

$$\mathrm{IFG}_{switch} \approx \mathrm{IFG}_{server}, \quad \Delta_{IFG} \approx 0$$

When $R_{enc} > \mu$:

$$\mathrm{IFG}_{switch} > \mathrm{IFG}_{server}, \quad \Delta_{IFG} > 0$$

Congestion onset is detected when:

$$Q^{(f)} > \theta_Q \quad \text{and} \quad \Delta_{IFG}^{(f)} > \theta_{IFG} \tag{2}$$

This joint condition distinguishes sustained congestion from transient jitter.

B. Profile Adaptation

Upon congestion, NICoLE selects a lower-capacity profile P_{k-1} from predefined set $\mathcal{P}$, prioritizing FPS and GoP stability while adapting spatial resolution to match service capacity. The selected profile remains consistent with GCC's bitrate constraint.

When congestion subsides ($Q^{(f)} < \theta_{low}$), the profile is gradually restored under stable pacing.

C. Transient L4S Steering

Packets are conditionally marked ECT(1) during transient congestion to enable temporary L4S queue admission [4], [11], [12]. Unlike persistent flow-level L4S marking, this mechanism is short-lived and aligned with the encoder adaptation interval.

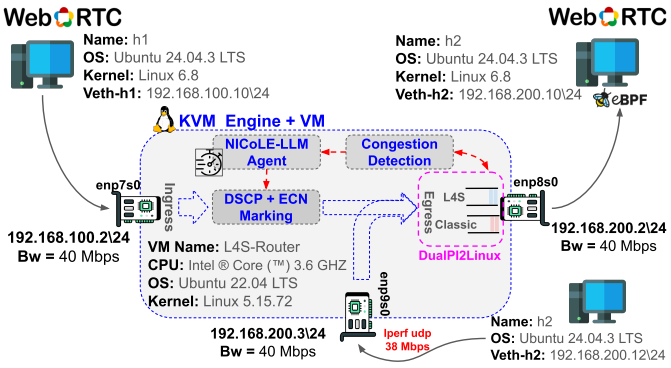


Fig. 3: Experimental testbed with an L4S-enabled router. The bottleneck link is shaped to 40 Mbps using `tc` with DualPI2.

D. Control Loop and LLM Design

NICoLE operates with decision interval $T \approx 320$ ms. RTP and queue features (FS, IFG_{server}, IFG_{switch}, Δ_{IFG}) are aggregated into structured inputs, and the LLM outputs a bounded action tuple $(P_k, L4S\_flag, DSCP_k)$ stored per flow and enforced in the dataplane.

Inputs are numeric and fixed-field; outputs are restricted to discrete actions. This design avoids free-form reasoning, ensures deterministic behavior, and prevents hallucination. Telemetry aggregation (∼500 ms) aligns observation and inference latency.

IV. Experimental Setup

A. Testbed Architecture

Fig. 3 shows the topology. Two WebRTC endpoints communicate through a KVM-based router acting as the in-network device. The router runs Ubuntu 22.04 (kernel 5.15) with DualPI2 [12], traffic shaping via `tc`, and the NICoLE control plane. Endpoints use GStreamer-based WebRTC with DSCP signaling via DataChannel. The bottleneck capacity is fixed at 40 Mbps.

B. LLM Model Selection

We evaluate compact LLMs (0.6B–8B parameters) under QLoRA fine-tuning and 4-bit quantization (GGUF) using `llama.cpp` on an 8-core 3.6 GHz CPU (8 GB RAM). Metrics include inference latency, throughput, and control accuracy. As shown in Fig. 4, larger models (e.g., 8B) incur latency (>1 s), while smaller models reduce accuracy. LLaMA-1B provides the best trade-off, achieving sub-400 ms latency with high control accuracy, consistent with the control interval.

C. Traffic Scenario

A controlled single-flow setup is used to isolate transient congestion. RTP streams are generated at fixed frame rates (30–120 FPS) over a 40 Mbps bottleneck.

Background UDP traffic (38 Mbps via `iperf`) is injected for 60 s within a 120 s run to induce deterministic oversubscription. Higher frame rates increase encoded demand and amplify congestion sensitivity.

TABLE I: Encoder profiles. FPS and GoP are fixed; only resolution is adapted.

Profile	Resolution	FPS	GoP
P0	3840×2160	Fixed	Fixed
P1	1920×1080	Fixed	Fixed
P2	1280×720	Fixed	Fixed
P3	640×360	Fixed	Fixed

D. Profiles and Encoder Configuration

The encoder supports predefined spatial profiles (Table I). FPS and GoP are fixed, and NICoLE adapts only spatial resolution, while GCC regulates bitrate.

E. Metrics

We evaluate perceptual, transport, and network performance:

- **Perceptual:** VMAF;
- **Temporal:** FPS stability;
- **Rate:** bitrate dynamics;
- **Queue:** occupancy and ECN behavior;
- **Buffer:** jitter-buffer occupancy and drops;
- **Pacing:** Δ_{IFG}.

We compare **GCC-only** with **NICoLE + GCC + L4S**.

To contextualize the LLM, a rule-based baseline using threshold logic can be defined. While effective in static settings, such rules require manual tuning and generalize poorly; NICoLE replaces this with a bounded adaptive decision model.

V. Experimental Results

A. LLM Inference Feasibility

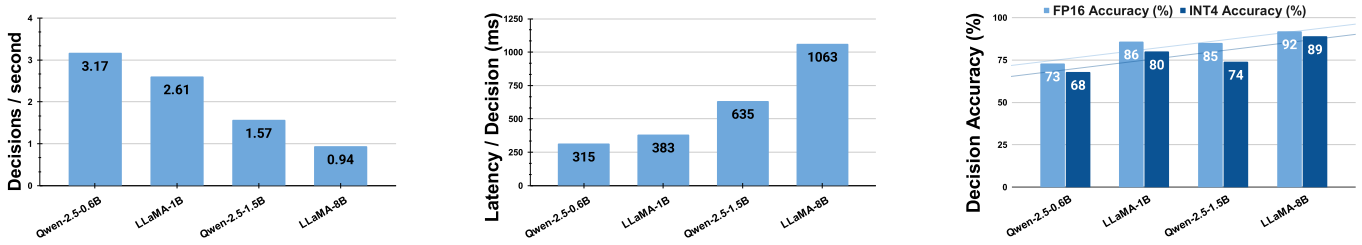
We evaluate whether in-network LLM control satisfies real-time constraints on commodity CPU hardware. Candidate models are tested under 4-bit quantization (GGUF) using `llama.cpp` on an 8-core 3.6 GHz CPU VM (8 GB RAM). Metrics include inference throughput, latency, and control accuracy.

Fig. 4 shows that smaller models provide higher throughput but lower accuracy, while larger models (e.g., 8B) incur excessive latency (>1 s). LLaMA-1B achieves 2.61 decisions/s with 383 ms latency and high accuracy, satisfying the 500–1000 ms control interval.

B. Flow Congestion Dynamics

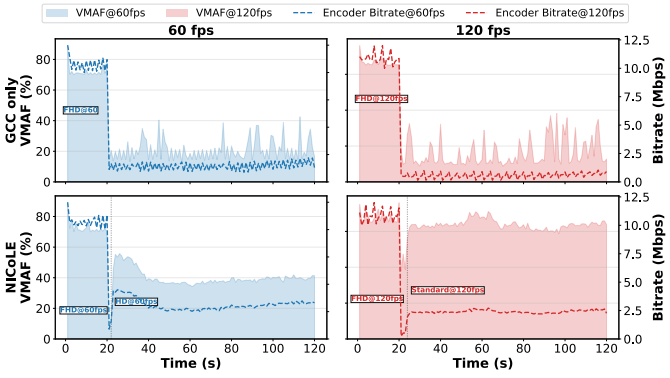
We evaluate transient behavior during a 60 s congestion window within a 120 s run under a 40 Mbps bottleneck. Experiments span 30–120 FPS. Fig. 5 shows representative time-series for 60 and 120 FPS, while Table II reports mean±std over five runs. Under GCC-only control, congestion is detected reactively via RTCP feedback, leading to delayed response, queue buildup, and bitrate oscillation. These effects intensify at higher frame rates due to shorter inter-frame intervals.

NICoLE detects congestion in-network using pacing distortion and queue state, enabling earlier intervention through profile adaptation and transient L4S steering.

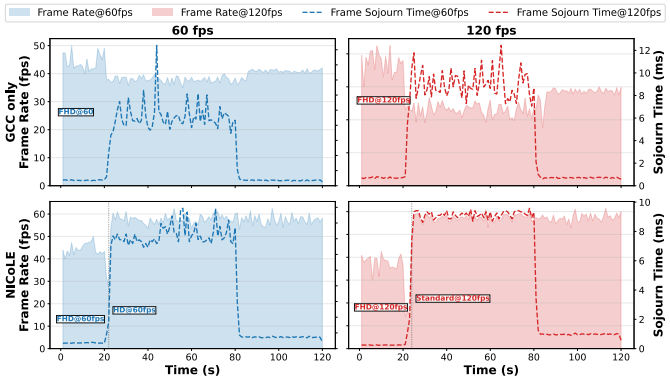


(a) Throughput (decisions/s) (b) Latency (ms) (c) Decision accuracy

Fig. 4: CPU-only LLM evaluation under 4-bit quantization. LLaMA-1B achieves the best latency–accuracy trade-off.



(a) VMAF and bitrate (60 FPS, 120 FPS)



(b) Frame rate and queue delay

Fig. 5: Time-series comparison under a 40 Mbps bottleneck. NICoLE stabilizes queue, pacing, and quality.

Table II shows that benefits increase with temporal density. At 60 FPS, NICoLE reduces peak queue delay by 61% (18.57→7.15 ms) and improves VMAF (20.83→40.15). At 90 FPS, quality improves by ≈18% with reduced delay variability. At 120 FPS, GCC collapses to 19.86 VMAF, while NICoLE maintains 63.56 VMAF with similar delay (11.49 vs 12.45 ms). At 30 FPS, GCC maintains stable performance and NICoLE provides limited benefit.

C. QoE Stability: Stall and Jitter

We evaluate stall ratio, frame loss, and jitter-buffer stability across 30–120 FPS.

TABLE II: Performance during congestion (mean±std over five runs).

FPS	Avg VMAF		Peak Delay (ms)	
	GCC	NICoLE	GCC	NICoLE
30	**29.35** ± 2.3	25.68 ± 1.75	7.42 ± 0.5	**3.06** ± 0.28
60	20.83 ± 2.08	**40.15** ± 2.12	18.57 ± 1.67	**7.15** ± 0.62
90	38.94 ± 4.1	**46.19** ± 3.5	12.32 ± 1.84	**9.60** ± 0.76
120	19.86 ± 2.38	**63.56** ± 5.08	12.45 ± 1.87	**11.49** ± 0.79

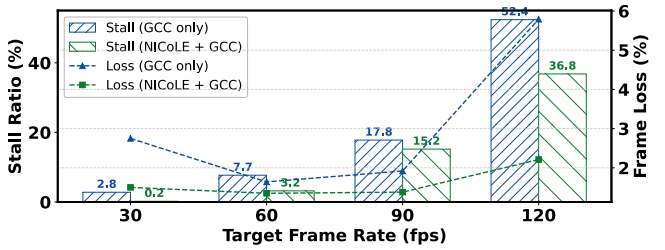


Fig. 6: Stall ratio and frame loss under congestion.

Stall ratio increases sharply with FPS under GCC, exceeding 50% at 120 FPS due to reactive bitrate control. NICoLE significantly reduces stalls at high frame rates by preserving temporal pacing through early profile adaptation.

Fig. 7 shows that GCC exhibits large playback variance at high FPS, while NICoLE maintains higher median FPS with lower dispersion. Improvements are most pronounced at 90 and 120 FPS.

These results confirm that NICoLE improves both perceptual quality and temporal stability, particularly under high-density workloads.

VI. Discussion: Cost–Gain Trade-off

An important question is when in-network LLM-assisted control justifies its computational cost. A simple feasibility condition is:

$$T_{inf} < T_{queue}$$

where T_{inf} is inference latency and T_{queue} is the effective GCC reaction time, including RTCP feedback and encoder response. From Sections V-A and IV-B, $T_{inf} \approx 383$ ms for LLaMA-1B, while T_{queue} ranges from 600 to 1500 ms [9], [18], [19]. This enables proactive intervention before significant queue buildup.

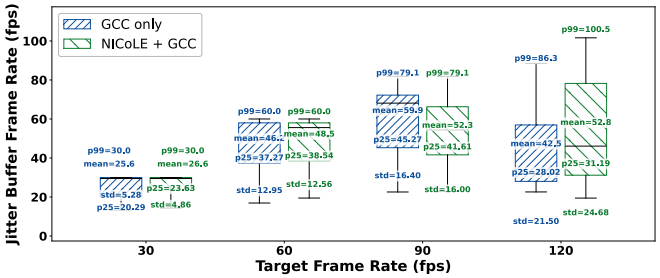


Fig. 7: Jitter buffer playback stability.

Empirical results show that benefit depends on temporal density. At 30 FPS, GCC alone maintains stable performance. At 90–120 FPS, shorter inter-frame intervals amplify queue instability, and early profile adaptation significantly improves quality and delay stability, indicating a transition region between 60 and 90 FPS.

LLM-based control should be interpreted cautiously. The decision problem is low-dimensional and can be approximated using rule-based or lightweight ML methods. The advantage of LLMs lies in flexibility across heterogeneous conditions without manual tuning. NICoLE therefore demonstrates conditional benefit rather than universal superiority.

A. Architectural Implications and Future Directions

Network-assisted codec adaptation improves temporal stability primarily under high-density workloads, but requires workload-aware deployment due to computational overhead.

Extending NICoLE to multi-flow environments introduces fairness and resource-sharing challenges. Executing transformer inference directly in programmable dataplanes remains infeasible at line rate, though emerging systems such as FENIX [20] and Taurus [21] suggest potential paths toward in-network ML acceleration.

B. Limitations

The evaluation focuses on a single-flow scenario to isolate transient dynamics. In multi-flow settings, interactions between L4S and Classic queues may affect fairness and stability. The controlled bandwidth environment simplifies telemetry patterns. The prototype uses CPU-based inference in a Linux VM; performance under higher concurrency and heterogeneous traffic remains future work.

VII. Conclusion

This paper presented NICoLE, an in-network agent that assists real-time RTP video streaming through proactive spatial adaptation and transient L4S steering. Results show reduced queue buildup, stall ratio, and playback instability under congestion, with the largest gains at high frame rates (90–120 FPS), where reactive bitrate-only control becomes insufficient.

The results highlight that LLM-assisted in-network control is conditionally beneficial: gains increase with temporal density and congestion sensitivity, while remaining limited in low-FPS or over-provisioned scenarios. This establishes a workload-aware model for deploying network-assisted streaming. Future work will extend NICoLE to multi-flow environments and explore tighter integration with programmable dataplane architectures.

Acknowledgment

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), FAPESP grant 2021/00199-8, CPE SMARTNESS.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3550

[2] J. Uberti and H. Alvestrand, "WebRTC Media Transport and Use of RTP," 2021.

[3] A. Shirmarz, F. L. Verdi, S. K. Singh, and C. E. Rothenberg, "From Pixels to Packets: Traffic Classification of Augmented Reality and Cloud Gaming," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, 2024, pp. 195–203.

[4] A. Shirmarz, M. N. Bragatto, F. L. Verdi, S. K. Singh, C. E. Rothenberg, G. Patra, and G. Pongrácz, "In-Network AR/CG Traffic Classification Entirely Deployed in the Programmable Data Plane: Unlocking RTP Features and L4S Integration," in *Proc. IEEE Int. Conf. Network Softwarization (NetSoft)*. IEEE, 2025, pp. 477–485.

[5] "Understanding Latency Tolerance in Cloud-Based VR Gaming," in *Proc. of the ACM Multimedia Conference*. ACM, 2025.

[6] F. Monaco *et al.*, "Latency Sensitivity in Cloud Gaming: An Empirical Study," *Computer Networks*, 2025.

[7] P. Karimi and M. Alizadeh, "Tight Loops, Smooth Streams: Responsive Congestion Control for Real-Time Video," vol. 139, pp. 9:1–9:0, 2026. [Online]. Available: https://arxiv.org/abs/2309.16869

[8] H. Meng, Y. Zhuang, Y. Noushirvani, X. Huang, and Z. Meng, "MAE: More Adaptive Video Encoder for Consistent Low Latency in High-Quality Real-Time Communication," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. Santa Clara, CA: USENIX Association, Apr. 2026. [Online]. Available: https://www.usenix.org/conference/nsdi26/presentation/meng

[9] H. Alvestrand and A. Zanella, "WebRTC Congestion Control: Requirements and Candidate Algorithms," 2020.

[10] K. Nichols and V. Jacobson, "Controlling Queue Delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[11] IETF, "Low latency, low loss, scalable throughput (l4s) architecture," 2023.

[12] K. D. Schepper, B. Briscoe, and G. White, "Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)," RFC 9332, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9332

[13] S. R. Pokhrel, D. Satish, J. Kua, and A. Walid, "Distilling Large Language Models for Network Active Queue Management," 2025. [Online]. Available: https://arxiv.org/abs/2501.16734

[14] Ericsson, "AI Agents in the Telecommunication Network Architecture," Ericsson AB, Tech. Rep., 2025.

[15] ETSI, "AI in the Evolution of Autonomous Networks," European Telecommunications Standards Institute (ETSI), Tech. Rep., 2025.

[16] G. Fairhurst and C. Perkins, "Operational Considerations for Streaming Media," RFC 9317, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9317

[17] S. Wenger, Y.-K. Wang, and T. Schierl, "RTP Payload Format for H.264 Video," 2011.

[18] S. Holmer and H. Lundin, "A Transport-Wide Congestion Control Mechanism for RTP," 2020.

[19] R. Jesup and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media," RFC 8836, Jan. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8836

[20] X. Gao, T. Li, Y. Zhang, Z. Wang, X. Zeng, S. Yao, and K. Xu, "FENIX: Enabling In-Network DNN Inference with FPGA-Enhanced Programmable Switches," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, ser. NSDI '26, 2026.

[21] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: A data plane architecture for per-packet ml," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, Feb. 2022, pp. 1099–1114.