Camel: Frame-Level Bandwidth Estimation for Low-Latency Live Streaming under Video Bitrate Undershooting

Liming Liu, Zhidong Jia, Li Jiang — Peking University
Wei Zhang, Lan Xie, Feng Qian — ByteDance Ltd.
Leju Yan, Bing Yan, Qiang Ma — ByteDance Ltd.
Zhou Sha, Wei Yang, Yixuan Ban — ByteDance Ltd.
Xinggong Zhang — Peking University

Abstract

Low-latency live streaming (LLS) has emerged as a popular web application, with many platforms adopting real-time protocols such as WebRTC to minimize end-to-end latency. However, we observe a counter-intuitive phenomenon: even when the actual encoded bitrate does not fully utilize the available bandwidth, stalling events remain frequent. This insufficient bandwidth utilization arises from the intrinsic temporal variations of real-time video encoding, which cause conventional packet-level congestion control algorithms to misestimate available bandwidth. When a high-bitrate frame is suddenly produced, sending at the wrong rate can either trigger packet loss or increase queueing delay, resulting in playback stalls.

To address these issues, we present Camel, a novel frame-level congestion control algorithm (CCA) tailored for LLS. Our insight is to use frame-level network feedback to capture the true network capacity, immune to the irregular sending pattern caused by encoding. Camel comprises three key modules: the **Bandwidth and Delay Estimator** and the **Congestion Detector**, which jointly determine the average sending rate, and the **Bursting Length Controller**, which governs the emission pattern to prevent packet loss.

We evaluate Camel on both large-scale real-world deployments and controlled simulations. In the real-world platform with 250M users and 2B sessions across 150+ countries, Camel achieves up to a 70.8% increase in 1080P resolution ratio, a 14.4% increase in media bitrate, and up to a 14.1% reduction in stalling ratio. In simulations under undershooting, shallow buffers, and network jitter, Camel outperforms existing congestion control algorithms, with up to 19.8% higher bitrate, 93.0% lower stalling ratio, and 23.9% improvement in bandwidth estimation accuracy.

CCS Concepts

• **Networks** → **Network algorithms**; • **Information systems** → **Multimedia streaming**.

Keywords

Low-latency Live Streaming, Congestion Control, Frame-level Control, Large-scale Deployment

WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792535

ACM Reference Format:
Liming Liu, Zhidong Jia, Li Jiang, Wei Zhang, Lan Xie, Feng Qian, Leju Yan, Bing Yan, Qiang Ma, Zhou Sha, Wei Yang, Yixuan Ban, and Xinggong Zhang. 2026. Camel: Frame-Level Bandwidth Estimation for Low-Latency Live Streaming under Video Bitrate Undershooting. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3774904.3792535

1 Introduction

As interactive live streaming has become a mainstream form of web-based content delivery [26], the demand for low-latency live streaming (LLS) has surged, making it a pivotal technology in today's web ecosystem [25]. An end-to-end LLS system typically consists of two core components: the upstream transmission from broadcasters to the relay server, and the downstream distribution from the server to end-users. The streaming latency, playback smoothness, and video quality are the primary concerns for LLS. To reduce the latency, more and more LLS providers are switching the upstream link to real-time protocols such as WebRTC or SRT [12, 27, 35].

However, we observed a surprisingly counter-intuitive phenomenon in one of the largest global video streaming platforms. As shown by real-world data across approximately 2 billion sessions in Figure 2, most uploaded live streams utilize only less than 80% of the estimated uplink bandwidth. Even so, the stalling ratio remains high: over 10% for more than 5% of broadcasters. And more surprisingly, in traces where the stalling ratio exceeds 10%, the video bitrate is typically further below the estimated bandwidth.

Why does the stalling ratio remain high even when bandwidth is not fully used? For convenience, we refer to the phenomenon where the actual sending bitrate is persistently lower than the target bitrate or the estimated available bandwidth as **undershooting**, and we measure its severity by the **R**atio of the video **E**ncoded-bitrate to its **T**arget-bitrate (ETR), because the target bitrate is typically aligned with the estimated bandwidth.

The reasoning chain is long but insightful. As shown in figure 1, the root cause lies in the nature of RTC video traffic itself, which violates the fundamental assumption of continuous backlogging that most congestion control algorithms rely on. When the sender always has enough packets waiting to be transmitted, the measured network signals are influenced only by the network conditions, and thus accurately reflect the available bandwidth. However, real-time encoders operate on a frame-by-frame basis, with frame sizes varying significantly over time depending on content complexity and frame types. While encoders aim to match the average bitrate to the estimated bandwidth rather than sustaining a constant high rate, their instantaneous bitrate fluctuations often cause the sender

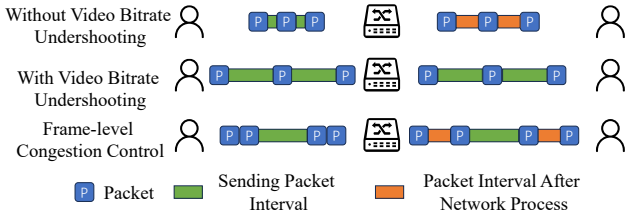


Figure 1: Without video bitrate undershooting, every packet can detect the network information; When video bitrate undershooting, feedback signals are also influenced by bitrate undershooting, causing estimation distortion; When using frame-level congestion control, some network information can be correctly measured within the individual frame burst.

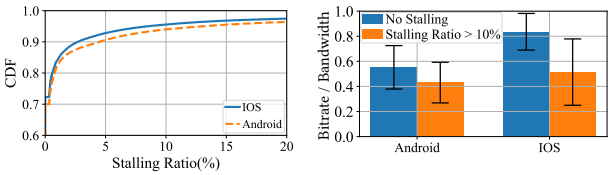


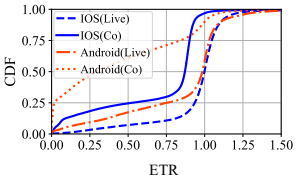
Figure 2: Distribution of stalling ratio and uplink bandwidth utilization among live streams' uploading links. Most live streams only utilize a moderate portion of the estimated uplink bandwidth. But the stalling ratio remains high. Additionally, in traces where stalling occurs, the video bitrate is typically further below the estimated uplink bandwidth.

to frequently drain the sending buffer. This leads to bitrate undershooting that distorts network feedback signals: observed packet loss rates and inter-arrival intervals are no longer purely reflective of network capacity, but are also shaped by application-layer dynamics. Such distortion results in inaccurate bandwidth estimation. When the encoder suddenly generates a burst of high-bitrate frames, the wrong bandwidth estimation could have significant consequences. If the sender transmits too aggressively, packets are dropped and transmission delays accumulate. If it transmits too conservatively, queueing delay increases. In both cases, once the delay exceeds the playback buffer, it leads to visible stalling events.

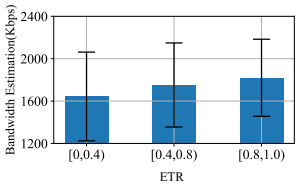
So can we design a congestion control mechanism that not only aligns with the encoder's natural and irregular frame-level burst pattern, but also leverages this pattern to more accurately estimate the available bandwidth? The answer is yes. **Our key idea is to transmit video data in a short, bursty pattern rather than a continuous stream, then leverage *frame-level network feedback* to capture the true network capacity, which is immune to the irregular inter-frame sending fluctuations.**

While promising, frame-level congestion control also introduces several unique challenges. First, identifying reliable frame-level indicators and accurately mapping them to bandwidth estimates under undershooting conditions is non-trivial. Second, detecting congestion at the frame level is difficult because delays of adjacent frames can interfere with each other. Third, controlling the length of each transmission burst is critical. Longer bursts provide more samples for stable bandwidth estimation, but excessive bursting can overwhelm small network buffers, causing packet loss.

To address these challenges, we present Camel, a novel frame-level congestion control algorithm tailored for LLS upstream video



(a) Distribution of actual-to-estimated bitrate ratio.



(b) Estimated bandwidth under different undershooting levels.

Figure 3: Distribution and impact of undershooting in RTC-based upstream video streams.

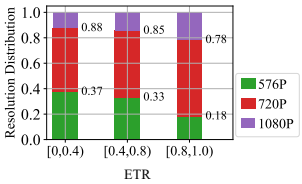


Figure 4: Lower ETR always leads to lower resolution.

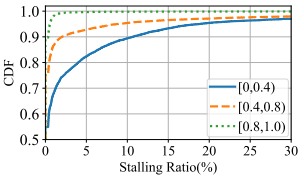


Figure 5: Lower ETR always leads to higher stalling ratio.

transmission. Camel consists of three key modules: the **Bandwidth and Delay Estimator** and the **Congestion Detector** jointly determine the average sending rate, while the **Bursting Length Controller** governs the packet emission pattern to avoid packet loss.

It is worth mentioning that Camel is motivated by the packet train concept and leverages a similar methodology to estimate network bandwidth and achieve frame-level congestion control. However, unlike prior studies on packet train [4, 8–10, 14, 16, 18, 23, 24, 28, 31–33], Camel introduces a novel congestion control algorithm tailored for LLS. In particular, it proposes the inflight-delay congestion detection method (§4.2), which effectively reduces congestion misjudgments, and designs a bursting length controller (§4.3) to prevent packet loss that introduced by frame-level bursting. It also proposes a bandwidth indicator in §4.1.

We have conducted extensive testing of our system in both large-scale real-world scenarios and controlled simulated environments. In real-world tests, our system was deployed on one of the largest live streaming platform, serving 250M users and 2B sessions across 150+ countries, demonstrating up to a 70.8% increase in 1080P resolution ratio, a 14.4% increase in media bitrate and up to a 14.1% reduction in stalling ratio. In simulated environments, we evaluated the algorithm's performance under video bitrate undershooting, shallow buffer and network jitter conditions. The results showed Camel significantly outperforms existing CCAs, achieving up to a 19.8% increase in bitrate, a 93.0% reduction in stalling ratio, and a bandwidth estimation accuracy improvement of up to 23.9%.

The remainder of this paper is structured as follows. In Section 2, we present the background and motivation behind our work. Section 3 provides an overview of Camel, including its key components and data flow. The detailed design of Camel is described in Section 4. We evaluate our system in Section 5, followed by a discussion of related work in Section 6. Finally, Section 7 concludes the paper.

2 Background and Motivation

2.1 Common Undershooting Causes Low QoE

As shown by real-world data collected from one of the largest global video streaming platforms, we observe that the *undershooting* behavior, where the actual sending bitrate is persistently lower than

the target bitrate or the estimated available bandwidth, is widespread across RTC-based upstream sessions. Figure 3(a) presents the distribution of the ratio between the actual encoded bitrate and the user's estimated upload bandwidth within 12-second intervals. The results show that more than 50% of streams exhibit varying degrees of undershooting. This common phenomenon can be attributed to several factors inherent to real-time video encoding. First, real-time encoders typically aim to match the average bitrate to the estimated available bandwidth rather than sustaining a constant high rate, since video content and frame type exhibits significant temporal fluctuations [13]: I-frames can be more than 10 times larger than P or B-frames, leaving the encoder with unused bandwidth for most non-I frames. Second, to maintain low latency and prevent excessive queuing, real-time encoders frequently lower the instantaneous bitrate, further contributing to persistent undershooting [12].

To understand why undershooting matters, we next examine its impact on bandwidth estimation. Figure 3(b) shows the estimated bandwidth across LLS streams grouped by different bitrate-to-bandwidth ratios. Since video bitrate ranges differ across resolutions, we use 720P streams as an example. The results indicate that as undershooting becomes more severe, existing congestion control algorithms (CCAs) tend to estimate bandwidth with a lower mean and higher variance. Given that the encoding bitrate is an inherent property of the video encoder, independent of transport-layer estimation [37], this correlation suggests that undershooting directly suppresses the sender's ability to probe for available capacity, leading to systematic bandwidth underestimation.

Such bandwidth underestimation has two direct consequences that jointly degrade user Quality of Experience (QoE). Figure 4 shows that the proportion of 720P+ videos drops significantly under lower bitrate-to-bandwidth ratios, while Figure 5 shows that the stalling ratio consistently increases in these cases. Together, these findings demonstrate that undershooting not only pervasively exists but also critically undermines both playback smoothness and perceived video quality in LLS systems. The reason is that, it prevents the sender from correctly pacing large video frames. When the encoder suddenly generates a high-bitrate frame, the sender faces two suboptimal choices. If it sends too aggressively, packets are dropped by the network and transmission delays accumulate. If it sends too conservatively, queueing delay increases. In both cases, once the delay exceeds the playback buffer, it leads to visible stalling events. Besides, under a persistently underestimated bandwidth, the adaptive bitrate (ABR) algorithm is misled to select lower video resolutions, resulting in visibly degraded visual quality [30].

2.2 Insight: Frame-Level Burst

The root cause of this unreliability lies in the temporal structure of RTC video traffic itself, which violates the fundamental assumption of continuous backlogging that most congestion control algorithms rely on. When the sender always has enough packets waiting to be transmitted, the measured network signals like throughput, delay, and loss, are influenced only by the network conditions. Traditional packet-level CCAs therefore perform effectively in bulk data transfer scenarios, such as file downloads. In these cases, the dynamics of the congestion window (*cwnd*) and the feedback-based adjustments of algorithms like BBR [2], CUBIC [11], or Copa [1] closely track the true available bandwidth. However, RTC traffic fundamentally breaks this assumption. Unlike file transfer or bulk streaming applications that always have enough packets to send, real-time encoders operate on a frame-by-frame basis. Each frame must be fully encoded before transmission, and frame sizes vary significantly over time depending on different content complexities and frame types. As a result, the sender transmits short, irregular bursts of packets followed by idle gaps. These encoder-induced silences frequently drain the sending buffer, causing bitrate undershooting that distorts network feedback signals: such observed packet loss rate and inter-arrival intervals are no longer purely reflective of network capacity, but also shaped by application-layer dynamics.

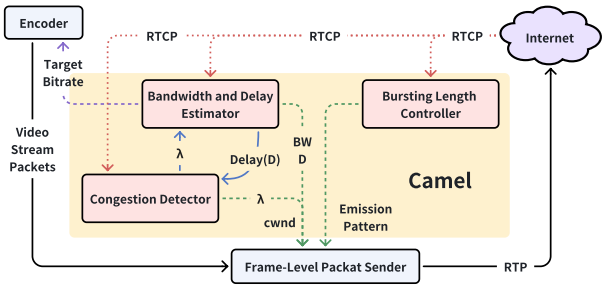


Figure 6: Camel's system architecture.

Motivated by this observation, we ask: can we design a congestion control mechanism that not only aligns with the encoder's natural and irregular frame-level burst pattern, but also leverages this pattern to more accurately estimate the available bandwidth?

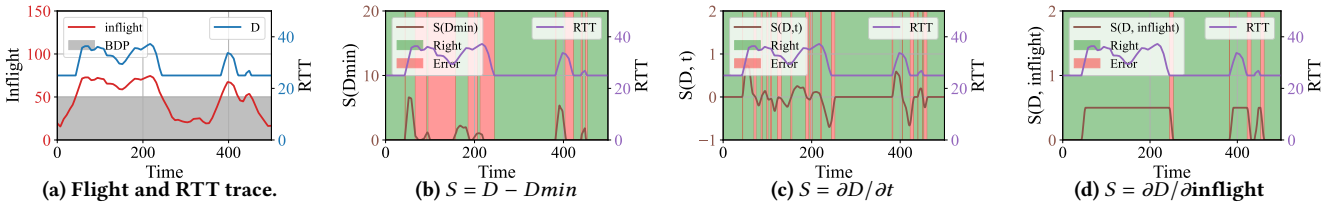
Our answer is yes. As illustrated in Figure 1, the key idea is to transmit video data in short, bursty segments rather than a continuous stream. We then leverage *frame-level network feedback* which aggregates network signals within a frame to estimate the available bandwidth. By operating on frame-level statistics, the estimation becomes immune to the irregular sending pattern or idle gaps between bursts, enabling the congestion controller to accurately capture the true network capacity, even under undershooting.

2.3 Challenges

While frame-level CCA appears promising, it also introduces several unique challenges that must be carefully addressed.

Challenge 1: Bandwidth and Delay Estimation from Frame-Level Feedback. Identifying reliable frame-level indicators and mapping them to accurate bandwidth estimates under undershooting conditions is a non-trivial task. The network feedback signals available at the frame level can be distorted by variations in frame size and encoding rate. Existing frame-level CCAs, such as SQP [24], which relies on frame transport bandwidth, and Pudica [32], which uses the bandwidth utilization ratio (BUR), are both affected by this issue. When a frame is smaller than the expected full-size frame, these algorithms compensate according to the undershooting ratio, which amplifies the statistical errors already present in small frames, leading to unstable or biased bandwidth estimation.

Challenge 2: Congestion Detection Based on Frame-Level Delay. Existing packet-level congestion signals have several drawbacks when applied to frame-level control. In particular, the delays



(a) Flight and RTT trace. (b) $S = D - Dmin$ (c) $S = \partial D/\partial t$ (d) $S = \partial D/\partial \textbf{inflight}$

Figure 7: The accuracy of different congestion signals, with $\partial D/\partial$inflight achieves the most accurate result.

of adjacent frames can easily interfere with each other, making it difficult to isolate congestion effects at the frame granularity.

Loss-based algorithms [11] often suffer from low throughput in the presence of physical packet loss and fail to effectively control latency in networks with deep buffers. MinRTT-probing-based algorithms [2], which use the minimum observed RTT as an estimate of propagation delay, are also unreliable in frame-level. A large tracking window for the minimum RTT cannot adapt to network jitter or dynamic background flows, while a small window risks producing non-convergent queueing delays once congestion arises. The delay gradient method [17], adopted in GCC, is more resilient to network jitter. However, it cannot actively drain persistent queueing delay once it stabilizes. Moreover, time-ordered delay gradients depend on assumptions such as consistent packet sizes and fixed pacing rates, which do not hold in frame-level congestion control, where both the packet rate and frame size vary substantially.

Challenge 3: Bursting Length Control. The buffer capacity of networks varies widely due to the heterogeneity of network infrastructures and carrier strategies [9]. As a result, the length of each transmission burst must be carefully controlled. A longer bursting length allows the algorithm to incorporate more packet samples within each burst, improving the stability of bandwidth estimation. However, excessive bursting can easily overwhelm small buffers, causing packet losses that typically emerging near the tail. Therefore, it is crucial to adaptively configure the maximum bursting length according to the network environment, balancing the accuracy of bandwidth estimation against the risk of buffer overflow.

3 Overview of Camel

To address the above challenges, we present Camel, a novel frame-level congestion control algorithm tailored for LLS upstream video transmission. A congestion controller must make two fundamental decisions: (1) how large the sending window (cwnd) should be, and (2) what packet emission pattern to use. Moreover, modern congestion control algorithms must ensure that the sending window closely matches the number of packets the network can accommodate, while rapidly responding to congestion signals [2, 11]. So the sending window is typically set according to this estimate:

$$\text{cwnd} = \gamma \cdot \hat{\text{BDP}}, \tag{1}$$

where $\hat{\text{BDP}}$ denotes the estimated bandwidth–delay product (BDP), and γ is a dynamic scaling factor that adapts to observed congestion to trade off utilization and queuing delay. In Camel, we decompose these decisions into three modules: the **Bandwidth and Delay Estimator** (for $\hat{\text{BDP}}$, §4.1) and the **Congestion Detector** (for γ, §4.2) jointly determine the cwnd, while the **Bursting Length Controller** (§4.3) governs the packet emission pattern. Each module directly targets one of the challenges identified in § 2.3.

Figure 6 shows the overall architecture of Camel: the **Bandwidth and Delay Estimator** continuously analyzes the incoming RTCP feedback to estimate both the available bandwidth and the delay. The estimated bandwidth is then mapped to the encoder's target bitrate, guiding rate control at the application layer, and simultaneously passed to the **Packet Sender** module. The estimated delay is forwarded to the **Congestion Detector**, which combines delay trends and RTCP feedback to derive a congestion signal that reflects real-time network status. Meanwhile, the **Bursting Length Controller** determines the appropriate burst length for the next transmission period based on the estimated network conditions. Finally, the Packet Sender uses the outputs from all three modules (estimated bandwidth, delay, congestion state, and burst length) to schedule packet transmissions from the encoder in a pattern that aligns with frame-level bursts while respecting network constraints.

4 Design of Camel

4.1 Bandwidth and Delay Estimator

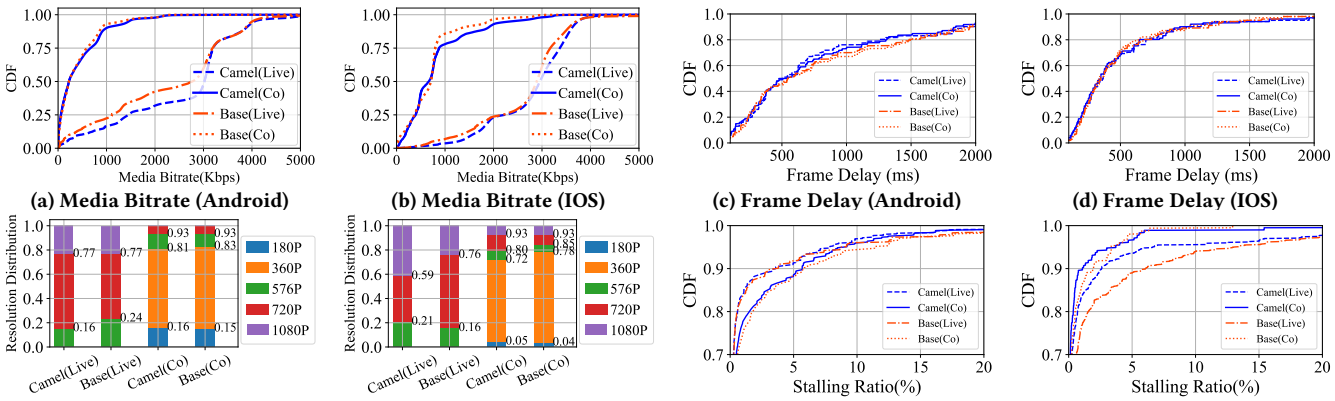
To address challenge 1, the bandwidth and delay estimator module takes RTCP feedback as input and outputs three key frame-level metrics: the bandwidth, the delay, and the bandwidth-delay product.

Frame-Level Bandwidth Estimation. Inspired by the packet train algorithm [9, 14] and Salsify [10], our fundamental idea is that by transmitting the packets of each frame in rapid succession, minimizing the inter-packet intervals, the receiver can accurately infer the available network bandwidth from the arrival rate of these packets. For a frame consisting of n packets, and these packets are numbered from 1 to n, the frame-level bandwidth B is defined as:

$$B = \frac{\sum_{i=2}^{n} S_i}{t_{recv_n} - t_{recv_1}}, \tag{2}$$

where S_i represents the size of the i-th packet and t_{recv_i} denotes its receiving time. Since this bandwidth is calculated independently for each frame, the sender is relieved from maintaining continuous packet backlogging at the application layer.

Frame-Level Delay Estimation. To mitigate the self-introduced delay distortion caused by burst transmissions, we define the frame-level delay D as the round-trip time (RTT) of the first packet in each frame. Unlike traditional pacing mechanisms that transmit packets at regular intervals, our frame-level approach sends all packets within a frame in a rapid burst. This bursty transmission pattern can temporarily saturate network buffers, introducing queuing delays that do not accurately represent the true network congestion state. Since only the first packet of a burst is unaffected by the queuing delay induced by its subsequent packets, its RTT serves as a more reliable indicator of the underlying network conditions.



(a) Media Bitrate (Android) (b) Media Bitrate (IOS) (c) Frame Delay (Android) (d) Frame Delay (IOS)

(e) Resolution (Android) (f) Resolution (IOS) (g) Stall Ratio (Android) (h) Stall Ratio (IOS)

Figure 8: Result of the large-scale A/B test, showing that Camel improves video QoE without compromising latency.

Frame-Level Bandwidth-Delay Product. With the estimated frame-level bandwidth and delay, we can further derive the frame-level bandwidth-delay product (BDP), leveraging similar principles from the classic congestion control model [2]:

$$\hat{BDP} = \text{avg}(B) \times \min(D), \tag{3}$$

where $\text{avg}(B)$ denotes the average frame-level bandwidth, reflecting the actual available network capacity, and $\min(D)$ represents the minimum observed frame-level delay, which approximates the round-trip propagation time (RTprop).

4.2 Congestion Detector

To address challenge 2 and enable robust congestion detection at the frame level, Camel uses the short-term trend of frame-level delay variations relative to the volume of in-flight data as its signal. Specifically, we compute the gradient of frame-level delay with respect to the number of bytes in flight. If this gradient exceeds a predefined threshold, congestion is inferred, as the gradient captures how queuing delay increases when the sender injects more data into the network. In the remainder of this section, we first justify the choice of this congestion signal through an experiment. We then describe how it is mapped to the cwnd scaling factor, and finally define the bandwidth value reported to the application layer.

Rationale for the Congestion Signal. To compare the accuracy of our proposed congestion signal against several commonly used detection methods, we generated a synthetic trace. We assumed a fixed network configuration with a bandwidth of 2 Mbps and a round-trip propagation delay (RTprop) of 25 ms, resulting in a bandwidth-delay product (BDP) of 50 Kb. A series of in-flight data values were then generated to simulate the bitrate fluctuations of an encoder, and the RTT of each packet was computed as follows:

$$\text{RTT} = \begin{cases} \text{RTprop} + \frac{\text{inflight} - \text{BDP}}{\text{Bandwidth}}, & \text{if inflight} > \text{BDP}, \\ \text{RTprop}, & \text{otherwise.} \end{cases} \tag{4}$$

This formulation follows a typical queuing delay model: RTT remains constant at the propagation delay when the in-flight data volume is within the network capacity, but increases linearly once the BDP is exceeded. Based on this model, we define the ground-truth congestion state as any moment when the RTT exceeds RTprop.

We compared our proposed inflight-based delay gradient signal, $S(D, \text{inflight}) = \partial D / \partial \text{inflight}$, with two commonly used alternatives: the MinRTT-based queuing signal, $S(D_{\min}) = D - D_{\min}$, and the time-based delay gradient, $S(D, t) = \partial D / \partial t$. All methods utilized a consistent 5-second observation window for fair comparison.

As shown in Figure 7, the highlighted regions indicate time intervals where each method correctly or incorrectly identifies congestion. For the other two signals, errors occur after the queue has built up, whereas our inflight-based delay gradient consistently achieves substantially higher overall accuracy than the baselines.

Mapping Congestion Signal to Window Scaling. To control the sending window based on the detected congestion, we set the initial value of γ to 1. When congestion is detected, γ is reduced by multiplying it by 0.95 to proactively decrease the congestion window size and mitigate potential queue buildup:

$$\gamma_{t+1} = \gamma_t \times 0.95, \tag{5}$$

If no congestion is detected, γ is reset to 1, allowing the cwnd to return to its full size and fully utilize the available bandwidth.

Congestion-Adjusted Bandwidth Reporting. To provide the application layer with a more accurate view of available network capacity, Camel reports a congestion-adjusted bandwidth to the application layer. The reported target bitrate is computed as:

$$\text{target bitrate} = \gamma \times \text{avg}(B), \tag{6}$$

where γ is the window-scaling factor derived from congestion detection and $\text{avg}(B)$ is the average frame-level bandwidth.

4.3 Bursting Length Controller

To address Challenge 3, Camel introduces the bursting length controller, which aims to maximize the burst length without causing packet loss. We infer the network buffer length from the observed packet loss rate because we make the following observation: for deep buffers, packet loss is primarily caused by physical factors, manifesting as similar packet loss rates across frames of varying sizes. In contrast, for shallow buffers, packet loss is mainly due to buffer overflow, resulting in higher loss rates for larger frames. Specifically, we cut frames into a fixed 2 KB intervals, and the packet loss rate is computed separately for each interval. The smallest loss rate of the smallest interval, L_0, serves as an estimate of the physical loss rate. The buffer length estimate M is updated every 5 seconds:

$$M_t = \begin{cases} M_{t-1} - 2\ \text{KB}, & \text{if } L_i > L_0 + 0.1 \\ M_{t-1} + 2\ \text{KB}, & \text{otherwise} \end{cases} \quad (7)$$

where L_i refers to the loss rate of the current interval. If the loss rate of the current interval surpasses 10% above L_0, M is decremented. Otherwise, M is incremented. If M_t has already reached its minimum value and the loss rate is still above L_0, the algorithm falls back to a GCC-based approach. Once the buffer length M is estimated, it is used to constrain the maximum frame length in periodic frame sending algorithm, mitigating the risk of buffer overflow.

5 Evaluations

In this section, we evaluated Camel through five key aspects: Large-scale A/B testing in real-world scenarios (§5.1); Robustness under video bitrate undershooting conditions (§5.2); Performance in weak network environments (§5.3); Component analysis to isolate and understand individual contributions (§5.4); and Convergence behavior when competing with background traffic (§5.5).

Setup: Our large-scale experiments were conducted on one of the world's largest video streaming platforms, which serves over 250 million users and 2 billion sessions across more than 150 countries. Due to the caution required for deploying new algorithms in a production environment, our large-scale A/B testing was limited to comparisons against the platform's existing congestion control algorithm, which is a BBR-based variant. All other experiments were performed in a self-built software simulation environment.

Baselines: In the simulation experiments, Camel is compared against six baseline algorithms: GCC [3], BBR [2], Copa [1], Pudica [32], SQP [24], and Salsify [10]. Due to Salsify's requirement for a functional encoder, it could not be integrated into our streaming system. Thus, we evaluated it using its open-source implementation. Because of the substantial differences in processing pipelines, Salsify's frame delay is not directly comparable and therefore omitted from our results. Moreover, as Salsify lacks an explicit bitrate target, we excluded it from the undershooting experiments.

Metrics: We evaluate Camel using four key metrics: *Media Bitrate*, *Frame Delay*, *Stalling Ratio*, and *Bandwidth Estimation Accuracy*. Media Bitrate quantifies the effective media transmission rate, reflecting both quality and efficiency. Frame Delay measures the end-to-end latency for a video frame. Stalling Ratio denotes the fraction of playback time spent in stalling, where stalling duration is defined as frame intervals exceeding 200 ms recorded by the server. Bandwidth Estimation Accuracy assesses how closely the bandwidth reported to the application layer matches the actual throughput, ensuring fair comparison across algorithms.

5.1 Large-Scale Real-World A/B Test

Overall, the large-scale A/B test demonstrates that Camel improves video quality and smoothness without compromising latency.

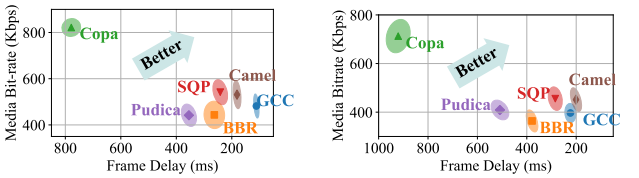
Media bitrate. As shown in Fig. 8(ab), Camel consistently achieves higher media bitrate than the baseline. For Android users, it improves the average bitrate by 11.9% in live streaming and 7.6% in co-broadcasting. For iOS users, the gains reach 3.5% and 14.4%.

Resolution. Fig. 8(ef) shows that Camel enhances video resolution by leveraging its higher bitrate. On Android, it increases the proportion of videos at or above 720P by 10.5%. On iOS, the share of 1080P videos in live streaming rises from 24% to 41%, and the proportion of at least 576P videos in co-broadcasting grows by 27%.

Stalling ratio. Fig. 8(gh) shows that Camel achieves a lower stalling ratio overall. For Android, the difference is minor, while on iOS, the average stalling ratio in live streaming decreases by 14.1%. In co-broadcasting, the average remains similar, but tail stalling events are notably reduced, improving user experience in extreme cases.

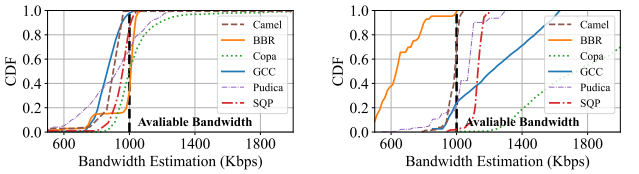
Frame delay. As shown in Fig. 8(cd), Camel maintains frame delay comparable to the baseline across both Android and iOS, while achieving significant improvements in bitrate and stability.

Padding bitrate. The baseline maintaining an average padding bitrate of at least 5 Kbps in live streaming and 25 Kbps in co-broadcasting, while Camel reduces padding bitrate to near zero because Padding occurs only for probing inactive connections.



(a) iOS (b) Android

Figure 9: Camel achieved a better trade-off between bitrate and frame delay in the simulation of undershooting scenario.



(a) Without undershooting (b) With undershooting

Figure 10: Camel can accurately estimate bandwidth in video bitrate undershooting scenarios.

5.2 Robustness under undershooting

Overall Performance. We evaluated the overall performance of all algorithms under video bitrate undershooting conditions. Two sets of bitrate undershooting traces were simulated for iOS and Android, following the distribution of the co-broadcasting scenario shown in Figure 3(a). The available bandwidth was capped at 1000 Kbps, and each algorithm ran for 150 seconds. Since no noticeable stalling was observed, we focused on media bitrate and frame delay.

As shown in Figure 9, Camel achieves higher bitrate than most baselines, except for Copa. In particular, it improves the average bitrate by 4.7% on Android and 19.8% on iOS compared to GCC. This improvement stems from Camel's ability to accurately estimate available bandwidth even when the encoder output undershoots. In contrast, BBR suffers from persistent bandwidth underestimation, while Pudica exhibits high volatility due to its assumption of stable frame sizes. Copa tends to overestimate bandwidth, resulting in excessive frame delays that are difficult to recover from promptly.

Bandwidth Estimation. We further examined the accuracy of bandwidth estimation under undershooting conditions. The available bandwidth was fixed at 1000 Kbps, and each algorithm ran for 90 seconds. Between 30 s and 60 s, the encoding ratio was manually reduced to 60% to emulate undershooting. We compared the

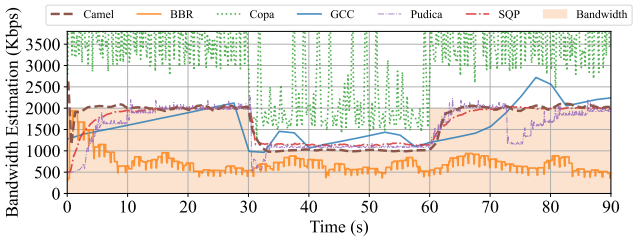


Figure 11: Camel can correctly follow bandwidth fluctuations under the video bitrate undershooting scenario.

estimated bandwidth during this period with that from the normal phase, as shown in Figure 10. Among the baselines, only the frame-level algorithms SQP and Pudica are capable of estimating bandwidth under undershooting. BBR exhibits severe underestimation due to reduced receiving rates, while delay-based schemes such as GCC and Copa overestimate bandwidth when no delay buildup is observed. In contrast, Camel achieves both higher accuracy and stability owing to its frame-level design, which remains robust even with varying frame sizes. Specifically, Camel achieves an average estimation accuracy of 98.9%, improving upon GCC by 23.91%.

Responsiveness. We evaluated the algorithms' responsiveness to bandwidth changes under undershooting. Each algorithm ran for 90 seconds: bandwidth was 2000 Kbps for the first 30 s, reduced to 1000 Kbps for the middle 30 s, and restored to 2000 Kbps for the final 30 s. Bandwidth estimates over time are shown in Figure 11. Frame-level algorithms, including Camel, SQP, and Pudica, correctly tracked bandwidth fluctuations. BBR, however, remained low due to consistently reduced receiving rates under undershooting. Delay-based algorithms, GCC and Copa, overestimated bandwidth and thus responded poorly to actual network changes.

5.3 Performance in weak network

Performance under real-world 4G/5G traces. We evaluated Camel on 4 real-world LTE traces from the public dataset [22], covering diverse network types and mobility conditions. Table 1 shows that Camel reduces stalling ratio by 13%–49% compared to other frame-level methods (SQP, Pudica, Salsify). Salsify suffers from low frame rate due to high encoder resource usage, Pudica assumes constant frame sizes, and SQP converges slowly, causing stalls during sudden bandwidth drops. Compared to GCC, Camel achieves up to 52% higher bitrate on 4G traces, as its packet-train-based design quickly reacts to bandwidth increases. Although BBR and Copa achieve high throughput and low stalling, Camel reduces frame delay by 12%–17%, avoiding periodic buffer-filling overheads.

Performance against network jitter. Figure 18 shows Camel under network jitter with 1000 kbps bandwidth. BBR, SQP, and Pudica rely on minRTT probing, misinterpreting transient jitter as persistent congestion, causing bandwidth underestimation, insufficient FEC transmission, severe stalling, and high frame delay. Copa mitigates jitter by filtering extreme RTTs, achieving higher bitrate than minRTT methods, but its conservative updates delay congestion resolution, leading to persistent stalls. GCC uses delay gradients and maintains low frame delay with minimal stalling. Camel further improves robustness by computing delay gradients over inflight-ordered frames instead of time-ordered packets, reducing jitter impact and achieving bitrate up to 94.9% higher than GCC.

Performance against packet loss. Figure 19 shows performance under packet loss with a 1000 Kbps bottleneck. As loss rate increases, all algorithms see bitrate drops due to reduced effective capacity. GCC and BBR severely underestimate bandwidth at high loss rates, impairing retransmission and FEC efficiency, and causing more rebuffering. SQP and Pudica overestimate bandwidth under high loss, also increasing rebuffering. At 50% loss, their stalling ratios are 4.53% and 2.62% higher than Camel, respectively.

Performance under shallow buffer cases. Figure 20 compares Camel with baseline algorithms for buffer sizes from 2KB to 10KB. At 2KB, SQP and Pudica lack packet samples for accurate bandwidth estimation, while Copa fails to detect congestion due to low queuing delay. These algorithms suffer persistent overshooting, causing high stalling and longer delay; packet loss further reduces media bitrate. BBR and GCC perform well by trading off bitrate and delay. Camel detects shallow buffers and falls back to GCC, achieving comparable performance and outperforming other frame-level methods. As buffer length grows, GCC degrades due to frequent mode switching, while SQP and Pudica only start estimating bandwidth accurately above 4KB and 6KB, respectively. Camel still provides superior QoE by optimally adjusting frame length.

5.4 Component Analysis

Bursting Length Controller. The effectiveness of the Bursting Length Controller is shown in Figure 12. For very shallow buffers (2KB), where packet train-based methods fail, the controller reduces burst length, enabling Camel to fall back to GCC. As buffer length increases, burst length scales accordingly, maximizing samples for Bandwidth Sampler while avoiding buffer overflow. At 10KB, bursts exceed 12KB, allowing most large frames to be sent fully, reducing frame delay and improving bandwidth estimation.

Figure 13 compares Bursting Length Controller with a fixed 20KB baseline. In a 2KB buffer, it achieves 1.79× media bitrate and reduces stalling by 67%. For 4–6KB buffers, it prevents unnecessary packet loss, reduces retransmissions and FEC, significantly improves QoE.

5.5 Convergence with Background Traffic

We evaluated Camel's congestion detection in multi-stream scenarios, using a common bandwidth of 1000 Kbps. Results showes that it achieves good fairness with various types of background traffic.

Inter-flow fairness. We evaluated the fairness among multiple Camel flows with the same RTT. A new Camel flow was introduced every 10 seconds. As shown in Figure 14, Camel successfully achieved fair bandwidth allocation and convergent latency.

Competitiveness with Inelastic Traffic. We evaluated Camel's ability to coexist with inelastic traffic, such as UDP, ensuring that it does not continuously compete for bandwidth unnecessarily. The experiment ran for 90 seconds, during which an UDP flow joint in. We collected the throughput and packet one-way delay. As shown in Figure 15, Camel effectively shared bandwidth with UDP while maintaining stable and controlled latency.

Competitiveness with Elastic Traffic. We then evaluated Camel's ability to coexist with elastic traffic like GCC, ensuring that it neither starves nor being starved when competing with low-latency algorithms. Camel ran for 90 seconds, during which a GCC flow was induced. Figure 16 shows the traces of throughput and RTT of

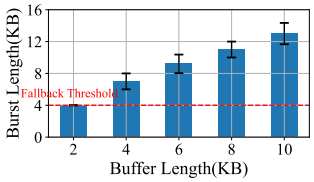
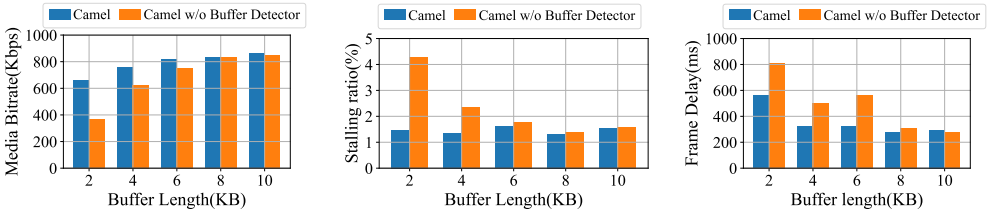


Figure 12: Camel's determined burst length for different network buffer lengths.



(a) Media bitrate **(b) Stalling ratio** **(c) Frame delay**

Figure 13: Camel's QoE improvement with Bursting Length Controller against shallow buffer.

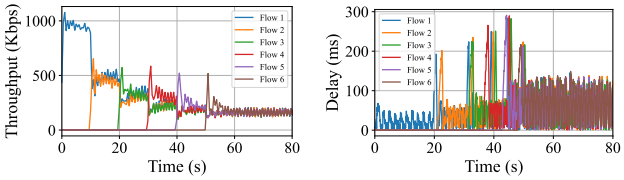


Figure 14: Camel can achieve good inter-flow fairness.

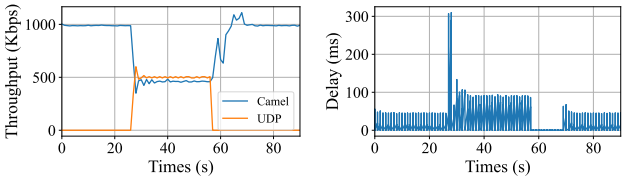


Figure 15: Coexisting with UDP.

the two flows, indicating that Camel successfully shared bandwidth with GCC while achieving controllable latency. When coexisting with buffer fillers like TCP Cubic, Camel does not aggressively compete for bandwidth, as excessive latency negatively impacts the experience of LLL streaming. While Camel is able to maintain a fair share of bandwidth under shallow network buffers, allowing TCP to quickly gain bandwidth, complete its data transmission, and exit the network can be an effective strategy in deep buffer scenarios. Figure 17 presents an experiment where a TCP CUBIC flow was introduced during Camel's video stream transmission. The results show that when the network buffer is set to 500 KB, Camel maintains a minimal transmission rate. However, when the network buffer is decreased to 100 KB, Camel achieves roughly equal bandwidth sharing with TCP CUBIC.

6 Related Works About Bandwidth Prediction in Undershooting Condition

Application-limited. Sammy [28] is a new adaptive bitrate (ABR) algorithm designed to smooth the video traffic, making it more friendly to internet applications. However, when traffic is smoothed, throughput-based algorithm struggle to accurately estimate bandwidth, placing Sammy in the same "Application-Limited" condition as our approach. To mitigate this, Sammy over-transmits traffic during initialization and subsequently adjusts the level of over-transmission based on specific algorithmic requirements. However, if the underlying CCA can effectively handle application-limited conditions, Sammy would have greater flexibility in selecting ABR algorithms while maintaining smooth traffic.

Adaptive Bitrate and Frame Rate. ABR and frame rate adaption are hot topics for LLS. Fugu [34] leverages a deep neural network trained on real-world data for robust throughput prediction. Sensei [36] improves ABR algorithms by evaluating users' quality sensitivity to different parts of the video. SODA [5] optimizes three key metrics: bitrate switching, bitrate, and stalling, while reducing algorithmic complexity to enhance deployability. ARTEMIS [29] dynamically configures the bitrate ladder for live streaming to adapt to time-varying video content. AFR [21] introduces adaptive frame rate control to improve the frame rate and resolution of video streaming. If CCAs can provide stable and accurate bandwidth estimation even under application-limited conditions, ABR and frame rate adaptation can be further optimized for improved performance.

Transport Layer Optimization. Beyond the design of CCA, there are several other optimization approaches at the transport layer. AUGUR [38] leverages multi-path transport over cellular networks to reduce long tail latency caused by Wi-Fi fluctuation in cloud gaming. Converge [7] enhances multi-path algorithms by incorporating video frame information, improving frame transmission integrity. Zhuge [19] accelerates the response of CCAs by proactively blocking ACKs on the access point (AP) based on the status of network queue. Hairpin [20] introduces a loss recovery mechanism that optimizes the combination of retransmissions and forward error correction (FEC) to meet deadlines while conserving bandwidth for edge-based interactive streaming.

Encoding and Transmission Interconnection. Several approaches have explored the coordination between encoding and transport layers to achieve better performance, often at the cost of increased deployment complexity. Swift [6] employs layered coding with neural video codecs to mitigate ABR issues caused by inaccurate network capacity estimations. BurstRTC [15, 16] predicts frame delay and QoE by modeling the distribution of frame sizes. Salsify [10] enhances encoder efficiency by integrating it with the transport layer through a custom encoder.

7 Conclusion

This paper addresses a critical bottleneck in designing upstream congestion control algorithms for existing low-latency live streaming systems, where video bitrate undershooting is prevalent. We propose Camel, a novel frame-level congestion control algorithm that mitigates the dependence on large or continuous data. Experimental results demonstrate that Camel provides accurate bandwidth estimation across diverse network conditions, while significantly improving media bitrate and reducing rebuffering in real-world deployments. This work does not pose any ethical concerns. This work was sponsored by the NSFC grant(62431017), ByteDance Grant(CT20241126107484). We gratefully acknowledge the support of Key Laboratory of Intelligent Press Media Technology. The corresponding author is Xinggong Zhang.

References

[1] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 329–342.

[2] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.

[3] 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.

[4] Dispersions Carry. [n. d.]. What Signals Do Packet-pair Dispersions Carry? ([n. d.]).

[5] Tianyu Chen, Yiheng Lin, Nicolas Christianson, Zahaib Akhtar, Sharath Dharmaji, Mohammad Hajiesmaili, Adam Wierman, and Ramesh K Sitaraman. 2024. SODA: An adaptive bitrate controller for consistent high-quality video streaming. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 613–644.

[6] Mallesham Dasari, Kumara Kahatapitiya, Samir R Das, Aruna Balasubramanian, and Dimitris Samaras. 2022. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 103–118.

[7] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2023. Converge: Qoe-driven multipath video conferencing over webrtc. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 637–653.

[8] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. 2004. Packet-Dispersion Techniques and a Capacity-Estimation Methodology. *IEEE/ACM Transactions On Networking* 12, 6 (2004), 963–977.

[9] Tobias Flach, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. 2016. An internet-wide analysis of traffic policing. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 468–482.

[10] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2018. Salsify:{Low-Latency} network video through tighter integration between a video codec and a transport protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 267–282.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[12] Debajyoti Halder, Prashant Kumar, Saksham Bhushan, and Anand M Baswade. 2021. FybrrStream: A WebRTC based efficient and scalable P2P live streaming platform. In *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–9.

[13] Xiangjie Huang, Jiayang Xu, Haiping Wang, Hebin Yu, Sandesh Dhawaskar Sathyanarayana, Shu Shi, and Zili Meng. 2025. ACE: Sending Burstiness Control for High-Quality Real-time Communication. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 1182–1198.

[14] Raj Jain and Shawn Routhier. 1986. Packet Trains–Measurements and a New Model for Computer Network Traffic. *IEEE journal on selected areas in Communications* 4, 6 (1986), 986–995.

[15] Zhidong Jia, Yihang Zhang, Qingyang Li, and Xinggong Zhang. [n. d.]. Tackling Bit-Rate Variation of RTC through Frame-Bursting Congestion Control. ([n. d.]).

[16] Zhidong Jia, Yihang Zhang, Qingyang Li, and Xinggong Zhang. 2024. BurstRTC: Harnessing Variable Bit-Rate of RTC through Frame-Bursting Congestion Control. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*. 213–214.

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

[18] Srinivasan Keshav. 1991. A control-theoretic approach to flow control. In *Proceedings of the conference on Communications architecture & protocols*. 3–15.

[19] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. 2022. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 193–206.

[20] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. 2024. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 907–926.

[21] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. 2023. Enabling high quality {Real-Time} communications with adaptive {Frame-Rate}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1429–1450.

[22] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand AK Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, et al. 2020. Lumos5G: Mapping and predicting commercial mmWave 5G throughput. In *Proceedings of the ACM internet measurement conference*. 176–193.

[23] Craig Partridge, Dennis Rockwell, Mark Allman, Rajesh Krishnan, and James Sterbenz. 2002. A swifter start for TCP. *BBN Technologies, Cambridge, MA, BBN Technical Report* 8339 (2002).

[24] Devdeep Ray, Connor Smith, Teng Wei, David Chu, and Srinivasan Seshan. 2022. Sqp: Congestion control for low-latency interactive video streaming. *arXiv preprint arXiv:2207.11857* (2022).

[25] Grand View Research. 2023. Asia Pacific Video Streaming Market Size, Share & Trends Analysis Report By Component (Hardware, Software, Services), By Solution (Internet Protocol TV, Over-the-Top), By Streaming Type (Live/Linear, Video On Demand), By Platform (Smartphones & Tablets Smart TV, Desktop & Laptop, Gaming Consol), By Service (Consulting, Managed Services, Training & Support), By Revenue Model (Subscription, Transactional, Advertisement, Hybrid), By End-use (Consumer, Enterprise), By Country, And Segment Forecasts, 2023 - 2030. https://www.grandviewresearch.com/industry-analysis/asia-pacific-video-streaming-market Accessed: YYYY-MM-DD.

[26] Sandvine. 2024. 2024 Global Internet Phenomena Report. https://www.sandvine.com/global-internet-phenomena-report-2024.

[27] Iván Santos-González, Alexandra Rivero-García, Tomás González-Barroso, Jezabel Molina-Gil, and Pino Caballero-Gil. 2016. Real-time streaming: a comparative study between RTSP and WebRTC. In *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer, 313–325.

[28] Bruce Spang, Shravya Kunamalla, Renata Teixeira, Te-Yuan Huang, Grenville Armitage, Ramesh Johari, and Nick McKeown. 2023. Sammy: smoothing video traffic to be a friendly internet neighbor. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 754–768.

[29] Farzad Tashtarian, Abdelhak Bentaleb, Hadi Amirpour, Sergey Gorinsky, Junchen Jiang, Hermann Hellwagner, and Christian Timmerer. 2024. {ARTEMIS}: Adaptive Bitrate Ladder Optimization for Live Video Streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 591–611.

[30] Truong Cong Thang, Hung T Le, Anh T Pham, and Yong Man Ro. 2014. An evaluation of bitrate adaptation methods for HTTP live streaming. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 693–705.

[31] Vinod M Vokkarane, Jason P Jue, and Sriranjani Sitaraman. 2002. Burst segmentation: an approach for reducing packet loss in optical burst switched networks. In *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No. 02CH37333)*, Vol. 5. IEEE, 2673–2677.

[32] Shibo Wang, Shusen Yang, Xiao Kong, Chenglei Wu, Longwei Jiang, Chenren Xu, Cong Zhao, Xuesong Yang, Jianjun Xiao, Xin Liu, et al. 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)*. 113–129.

[33] David Wei, Pei Cao, Steven Low, and Caltech EAS. 2006. TCP pacing revisited. In *Proceedings of IEEE INFOCOM*, Vol. 2. Citeseer, 3.

[34] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 495–511.

[35] Wei Zhang, Tong Meng, Xianhua Zeng, Wei Yang, Changqing Yan, Chao Li, Chenguang Li, Feng Qian, Junfeng Yang, Lei Zhang, and Zhi Wang. 2024. Harnessing WebRTC for Large-Scale Live Streaming. In *Proceedings of the ACM SIGCOMM 2024 Conference*.

[36] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. 2021. {SENSEI}: Aligning video streaming quality with dynamic user sensitivity. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 303–320.

[37] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. 2019. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[38] Yuhan Zhou, Tingfeng Wang, Liying Wang, Nian Wen, Rui Han, Jing Wang, Chenglei Wu, Jiafeng Chen, Longwei Jiang, Shibo Wang, et al. 2024. {AUGUR}: Practical Mobile Multipath Transport Service for Low Tail Latency in {Real-Time} Streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1901–1916.

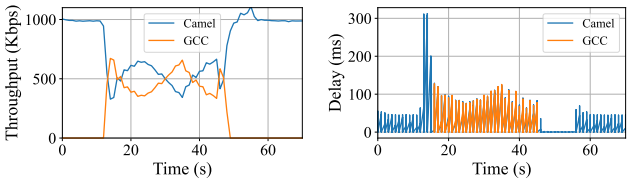
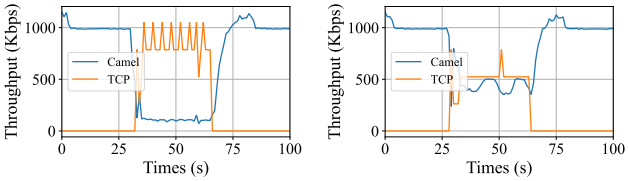


Figure 16: Coexisting with GCC.

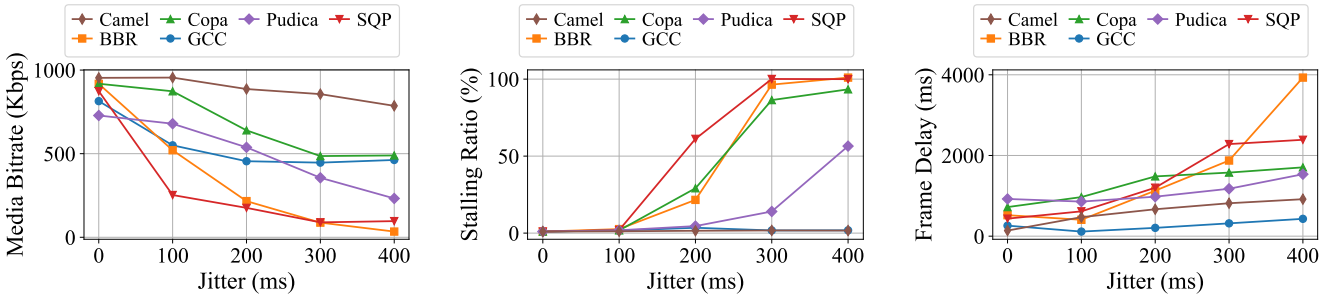


(a) $buffer = 500KB$. **(b)** $buffer = 100KB$.

Figure 17: Coexisting with TCP.

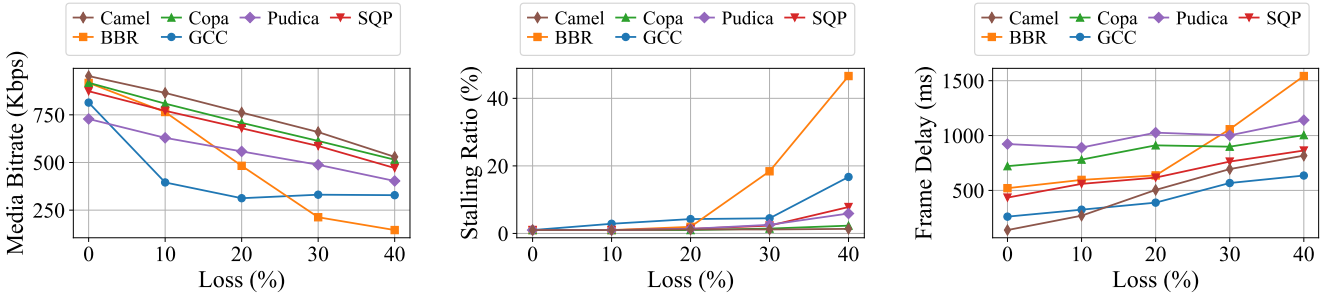
Table 1: Comparison of QoE over real-world LTE traces.

	4G_walking_50167			4G_driving_50187			5G_walking_90			5G_walking_109		
	Bitrate (Kbps)	Delay (ms)	Stall (%)	Bitrate (Kbps)	Delay (ms)	Stall (%)	Bitrate (Kbps)	Delay (ms)	Stall (%)	Bitrate (Kbps)	Delay (ms)	Stall (%)
Camel	1606.61	1014.57	6.35	2010.09	183.69	0.30	2030.23	135.34	1.42	1981.05	546.51	4.99
Copa	1697.66	997.89	6.78	2035.74	268.16	0.00	2042.70	185.39	1.36	2052.34	652.03	5.05
BBR	1668.18	1167.29	10.92	2014.49	195.75	0.29	2028.53	213.41	1.34	2042.48	687.53	4.84
GCC	1054.97	704.40	6.05	1624.72	139.36	1.02	2037.92	153.65	1.41	1958.70	606.71	5.40
SQP	1419.80	914.53	9.72	1954.26	188.76	2.41	1979.38	187.53	2.67	1860.21	634.29	8.61
Pudica	1535.89	1037.90	7.73	1930.93	275.97	1.20	1946.38	231.36	3.65	1884.94	603.07	9.03
Salsify	1504.33	/	12.69	2041.93	/	2.03	2059.46	/	1.56	2025.78	/	9.59



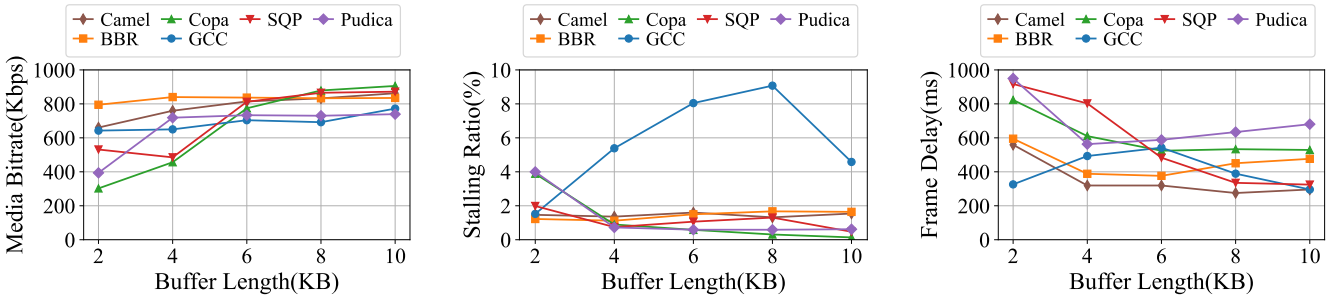
(a) Media bitrate. **(b) Stalling ratio.** **(c) Frame Delay.**

Figure 18: Performance under jitter.



(a) Media bitrate. **(b) Stalling ratio.** **(c) Frame Delay.**

Figure 19: Performance under loss.



(a) Media bitrate (b) Stalling ratio (c) Frame delay

Figure 20: Camel can maintain high bitrates, low stalling, and low frame delay under shallow buffer conditions.