USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

MAE: More Adaptive Video Encoder for Consistent Low Latency in High-Quality Real-Time Communication

Hua Meng, Yufan Zhuang, Yasna Noushirvani, Xiangjie Huang, and Zili Meng, *Hong Kong University of Science and Technology*

https://www.usenix.org/conference/nsdi26/presentation/meng

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

MAE: More Adaptive Video Encoder for Consistent Low Latency in High-Quality Real-Time Communication

Hua Meng, Yufan Zhuang, Yasna Noushirvani, Xiangjie Huang, Zili Meng
Hong Kong University of Science and Technology

Abstract

Real-time communication (RTC) is integral to modern digital life. However, high-quality RTC services still experience tail latency spikes. A primary cause of this issue is the sender's slow adaptation to network fluctuations – when network bandwidth drops, the sender needs to quickly reduce the sending rate to avoid bufferbloat. Existing solutions focus on accelerating reactions in the network, transport layer, or sender-side buffer management, but often overlook a critical component: the reaction of the video encoder. The encoder serves as the content source, where slow convergence to new bitrates can still result in bufferbloat after the encoder. With the increasing video bitrate in high-quality RTC, our measurement shows that the encoder's reaction is critical. To address this limitation, we propose **M**ore **A**daptive **E**ncoder(MAE), a framework that enables encoders to dynamically adapt to network changes with finer-grained network information. On the encoder side, MAE adaptively adjusts the internal encoding parameters to quickly converge to the target bitrate without affecting the video quality. On the network side, MAE probes the internal state of current congestion control algorithms to preemptively react to potential bandwidth drops, without waiting for the late, backpressured bitrate updates. Trace-driven emulation and real-world experiments demonstrate that our solution, MAE, reduces stall rates by 86.2% while maintaining superior visual quality compared to the state of the art.

1 Introduction

Real-time communication (RTC) has become an essential part of modern digital life, such as videoconferencing, cloud gaming, and virtual reality. Given its widespread adoption, reducing stalls in RTC would yield significant benefits for a large portion of users [24, 27]. However, with the increasing demand of high-quality RTC services, such as cloud gaming or VR, where the bitrate can be 30 to 100 Mbps [22, 30], the current stall experiences are still far from satisfactory due to the occasional increase in the tail latency. As well studied in the previous literature, one of the root causes of stalling is that the sender's sending rate cannot quickly react to the network fluctuations.

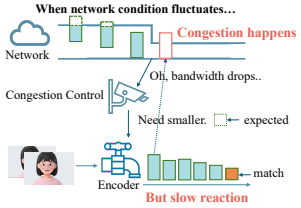


Figure 1: The encoder's slow response to bandwidth changes will lead to congestion and high latency.

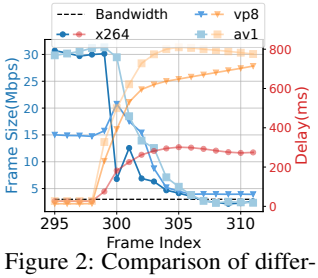


Figure 2: Comparison of different codecs' reaction time when bandwidth drops and simulated delay.

These network fluctuations are very common even in the most advanced wireless access networks such as WiFi-6 and 5G: network capacity can drop by 50× in 200 ms with a non-trivial frequency of 1% [28]. If the sender's sending rate cannot adapt to the new network capacity quickly, it will lead to detrimental accumulated latency, a.k.a. bufferbloat. For instance, if the sender delays adapting its bitrate by just one RTT after a 10× bandwidth drop, 9× RTT of latency accumulates immediately. This problem grows increasingly serious as video bitrates rise – a larger range for bandwidth to decrease amplifies the latency caused by the slow adaptation. Therefore, numerous research efforts have been devoted to accelerating the reaction, from router feedback [28], congestion control [3], and sender buffer management [21].

However, a critical but largely overlooked component in the video streaming pipeline is the reaction of the video encoder. For the end-to-end latency in real-time video streaming, only by reducing the bitrate of the video encoder can the latency ultimately be under control – this is because video encoder is the source (generator) of the content, as illustrated in Figure 1. For the acceleration of reaction at other components, it will help to mitigate the latency after that component. Yet, the accumulated bufferbloat simply shifts from downstream to upstream, still resulting in a high end-to-end delay during the fluctuation. Unfortunately, when the target bitrate changes,

the current video encoder cannot quickly converge to the new bitrate. It not only relies on a slow backpressure mechanism from the congestion control or rate control, but also suffers from the internal quality control mechanisms inside the encoder. We tested with a variety of different encoders. As illustrated in Figure 2, for commercial off-the-shelf encoders such as libx264 or libav1, it takes five frames (more than 100 ms) to converge. When the bitrate is reduced by a factor of 10, this will result in a huge bufferbloat, leading to severe stalls of up to 800 ms.

There have been efforts to make the encoder strictly follow the bitrate, but always at the cost of quality. For example, frame dropping can quickly address overshoot issues [16, 40], but the loss of consecutive frames can lead to severe stalls. Another approach involves strictly limiting frame size to prevent overshooting the target bitrate, such as constant bitrate (CBR) [35] and scalable video codec (SVC) [36]. But this strategy will significantly reduce the compression efficiency when the target bitrate is not fluctuating, therefore hardly adopted in practice [12].

Our key insight is this: for consistent low latency in real-time communications, *video encoders must be able to quickly adapt to network fluctuations*, rather than slowly converge to the new bitrate derived from the downstream congestion control. However, as discussed above, it is challenging to maintain the video quality and strictly follow the target bitrate at the same time.

Therefore, we want to make the encoder adaptive – it still maintains the high video quality when the target bitrate is stable, and switches to the mode of strictly following the bitrate when the bandwidth drops. To this end, we propose MAE (More Adaptive Encoder), a framework to quickly adapt to network bandwidth change, achieving low tail latency and stall ratio while maintaining the overall quality. Since the bandwidth only drops in a small portion of the time, the overall video quality will hardly be affected. Yet, the tail latency, which is mostly caused by the slow reaction when bandwidth drops, can be well controlled.

Translating this straightforward idea into practice poses two core challenges. The first is how to adjust it without compromising quality. Unlike directly modifying the encoder's target bitrate (which drastically degrades quality by capping overall output), we regulate the bitrate overshooting of the encoder without slashing the target rate. We, therefore, adaptively adjust the size of the encoder's internal virtual buffer, an internal encoding mechanism that manages the number of remaining bytes a video frame can use. We reduce the buffer size when a potential bitrate drop is detected, and quickly restore the buffer to its original size once the congestion has been mitigated.

The second challenge is determining when the encoder should adapt, or how to identify that a network drop is occurring. Passively waiting for the update of the target bitrate from the rate control is insufficient – when bandwidth drops, even the delay of one video frame can result in a drastic bufferbloat. Therefore, to mitigate the bufferbloat, we need to ensure that the encoder can immediately converge to the new bitrate when the bitrate drops. We choose to probe the internal states of congestion control algorithms (CCAs). Before CCA reduces the sending rate, some internal states must have already increased, just not enough to trigger the rate reduction (e.g., due to a moving average or not reaching a threshold). MAE probes those internal states to get prepared for the potential rate reduction. Note that CCAs have already done their best in reacting at the earliest time – those mechanisms are to denoise the measurements since the cost of mistakenly reducing the rate is also high [1]. On the contrary, getting prepared for a potential rate decrease in the video encoder is not too costly – the false positive can be tolerated since it only affects a small portion of frames. This bridges the gap between CCA's stability and the encoder's timely reaction.

We implemented MAE in WebRTC [6] and conducted both real-world traces driven emulation and real-world experiments to validate its effectiveness. Results demonstrate that, compared to the status-quo WebRTC, MAE reduces the stall rate by 86.2% while keeping overall quality degradation within 2.1%.

Our main contributions are summarized as follows:

- We highlight the necessity of encoders adapting dynamically to network conditions.
- We identify key challenges in developing MAE and propose a novel approach to address these challenges.
- We implemented MAE [1] in WebRTC and conduct extensive performance evaluations via trace-driven emulations and real world experiment, validating its practical effectiveness.

2 Background

2.1 Tail Latency in RTC: Growing Importance and Attention

Real-time communication (RTC) applications have become essential in modern digital life. Due to their widespread adoption, both academia and industry have focused significantly on optimizing a key metric for RTC quality of experience (QoE): *tail latency* (i.e., the higher percentiles of latency distributions). High tail latency often leads to stalls in the system. It has been widely acknowledged that the tail latency is important to the user experience [5, 17, 26, 37, 38]. For instance, with a 99.9th percentile tail latency, the video stream will experience that latency once every 1000 frames, usually no more than 30 seconds. These cumulative effects illustrate why even "rare" latency spikes have a greater impact than stable average latency during long sessions.

This can also be validated from a notable trend in recent years: as shown in Figure 3, the attention of the community has shifted from lower tail latency percentiles (e.g.,

[1] Code Repository: https://github.com/hkust-spark/sparkrtc

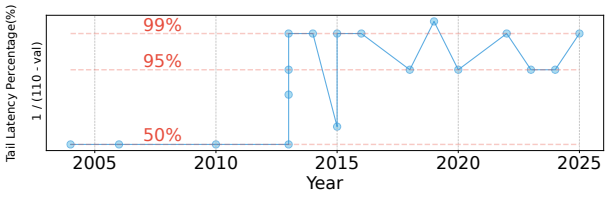


Figure 3: The focus of tail latency percentile over the last 20 years in NSDI and SIGCOMM. There has been an increasing focus on the 99th and 99.9th percentile in recent years.

the average or median tail latency) to increasingly higher targets, culminating in a focus on the 99.9th percentile. [2, 4, 10, 11, 13, 14, 16, 19, 21, 23, 25, 28, 32, 33, 39, 42, 43]. As RTC applications become more immersive and mission-critical, tolerance for tail latency spikes narrows. Consequently, controlling tail latency is a fundamental requirement for effective RTC systems.

2.2 The Root Cause of Latency Spikes: Reaction

The increase of tail latency often comes from the buffer in the delivery pipeline of RTC. In the end-to-end delivery pipeline of RTC, there are a series of buffers between every two components. For example, as shown in the simplified architecture in Figure 4, when a frame has been encoded, it will be put into the buffer before the packets are allowed to be sent out by CCA or rate control, such as GCC [7]. Although each individual component can also contribute to the end-to-end delay (e.g., video decoder or wireless channel), it rarely contributes at the magnitude of hundreds of milliseconds [30].

The buffer gets accumulated, a.k.a. bufferbloat, due to the mismatch between the arrival rate and outgoing rate of the buffer. For example, when the downstream network capacity drops, if the sender still sends at the original rate, the excessive packets will be blocked in the router buffer. The time to drain the buffer is always multiple times of the reaction time, since the draining rate (the dropped one) is lower than the filling rate (the original one) [28]. Going further, when the buffer is full, packets (or frames) can also be dropped, which will further result in a long end-to-end delay. Given the recent study [20] that up to 88% of RTC packet loss is driven by congestion, the consequence will only be drastic.

Existing solutions have tried to control the buffer at the downstream, yet the bufferbloat will shift to the upstream rather than disappear. For example, end-network coordination helps the sender to know the network conditions earlier [15, 18, 28] and sophisticated rate control algorithms aim at estimating the sending rate more accurately [3, 7]. They help to alleviate the bufferbloat on the routers, but since now the socket only allows a small amount of traffic to go out, the data will start to accumulate in the buffer at the sender. Subsequently, there have been research efforts in managing the sender buffer (socket buffer, pacing buffer) [9, 21]. Nonetheless, the bufferbloat keeps moving upstream to the video codec since the video codec needs some time to react

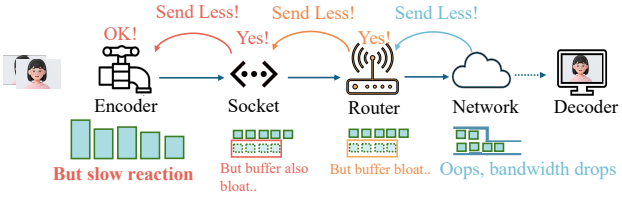


Figure 4: The demonstration of reasons that lead to stalls.

(§3.1). Therefore, as shown in Figure 4, only when the encoder quickly adapts to the new bitrate can the bufferbloat be fully eliminated from the pipeline. Figuratively, only turning off the first water tap can we fix the water overflow in a pipe.

The latency spikes will be amplified with the recent pursuit of high bitrate in RTC applications such as VR/AR and cloud gaming. For example, the video bitrate goes up from hundreds of kbps in traditional video chats to tens of Mbps. Yet, when network fluctuates, the available bandwidth might still decrease to hundreds of kbps [28]. The increased dynamic range of bitrate makes the bufferbloat issue more severe.

2.3 Existing Solutions

There have been some efforts, intentionally or unintentionally, to make the video encoder adaptive. However, it always comes at the cost of video quality.

Drop some data when overshooting. One straightforward idea to control the bufferbloat when the encoder has not converged to the target bitrate is just dropping the excessive data. For example, scalable video coding (SVC) [36] and loss-resilient codecs [8] allow a portion of a frame to be dropped while the frame remains decodable. Other works propose to wholly skip the oversized frames [16, 40] until the frame size is low enough to send. However, all these methods sacrifice the video quality to maintain its ability of partial decoding, or significantly increase the complexity of the encoder in a non-standard way. For example, a 4-layer SVC coding method demands 2.5 × more bits per pixel than its H.265 counterpart [12]. Therefore, it is difficult to deploy them in the wild at scale.

Strictly follow the bitrate all the time. Another method is to force the encoder to generate the video frames that immediately satisfy the target bitrate, such as constant bitrate (CBR) [35]. However, this reduces the flexibility of the video encoder in encoding dynamic content. The reason why the encoder needs a long time to react is that the contents are always dynamic – some complex and some simple. Flexibly allocating bits to different contents can maximize the encoding efficiency. CBR, however, significantly reduces such encoding efficiency by up to 18% by taking away the flexibility [21].

3 Motivation and Challenges

3.1 Motivation: Slow-Reacting Encoder

Observation: The root cause of tail latency spikes lies in slow-reacting encoders. Our analysis led to a clear conclu-

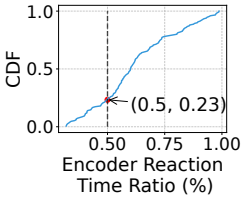


Figure 5: Encoder reaction time / Overall reaction time when bandwidth drops.

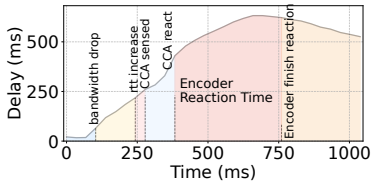


Figure 6: Details of a single bandwidth drop reaction.

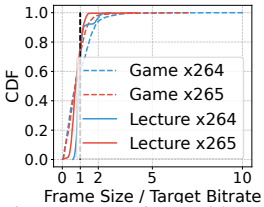


Figure 7: Even in a stable target bitrate, encoder outputs still fluctuate

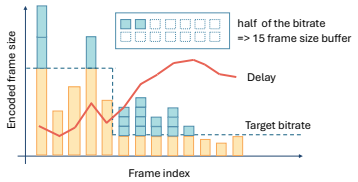


Figure 8: How the virtual buffer works in the encoder.

sion: the majority of tail latency spikes stem from the encoder's slow reaction to CCA-recommended bitrate changes. To measure the reaction time of existing RTC systems, we executed WebRTC over five simulated bandwidth drop traces (10→1 Mbps, 10→2 Mbps, 10→5 Mbps, 5→1 Mbps, 2→1 Mbps). Using the output logs, we extracted three key timestamps: when the CCA first detected a potential drop, when the CCA initiated a reaction, and when the encoder consistently operated below the target bitrate for two consecutive frames. We define the overall reaction time as the total duration spanning these three key timestamps. Based on this, we calculate the ratio of the encoder's reaction time to the total reaction time. The results, presented in Figure 5, show that over 77% of the total reaction time to bandwidth drops is attributed to the encoder's response latency.

More specifically, Figure 6 breaks down the time overhead of each component in a single test trial, enabling us to compare the response times of network components and the encoder to a bandwidth drop. For instance, when bandwidth drops from 10 Mbps to 1 Mbps, the CCA detects the trend and initiates a reaction in only 140 ms. In contrast, the encoder requires 377 ms to converge—accounting for 65.3% of the total reaction time—and this prolonged encoder response leads to significant latency accumulation.

Why does the current encoder take a long time to adapt? As noted in §2.3, current encoders are designed to allow temporary overshoots of the target bitrate for dynamic contents. As shown in Figure 7, even under a stable target bitrate, 19%–35% of frames still overshoot. This causes transient increases in latency, but does not lead to latency accumulation. Encoders implement this feature with a virtual buffer regulating these deviations to keep long-term output aligned with the target. This buffer is typically sized as a fixed fraction of the current target frame size – for example, often set to 50% in libx264. In scenarios with a stable target bitrate, this configuration works well: the buffer absorbs minor overshoots without causing issues, as the network can accommodate small, transient spikes.

The problem emerges during bandwidth drops. Consider a typical configuration: the buffer is sized to 50% of the target bitrate, which—for a 30fps stream—translates to a headroom equivalent to 15 frame size. As shown in Figure 8, when bandwidth drops, the buffer does adjust to 50% of the new, lower target bitrate, but this still leaves a headroom of 15 frame sizes relative to the reduced bandwidth. Besides, encoders predict frame sizes based on past content complexity and bit usage, leading to temporal coherence in frame sizes—they adjust gradually rather than abruptly. This gradualism, combined with the 15-frame buffer headroom, means that during a bandwidth drop, successive frames continue to consume most of the buffer's capacity. Over time, this depletes the buffer's margin, causing the encoder's output to overshoot the new target bitrate and exacerbate delays.

Why does a slow-reacting encoder become the root cause? First, the network layer has reached practical optimization limits. While CCAs typically exhibit a reaction time of 100 – 150ms due to the measurement window and reaction threshold, further reducing this window is not feasible. Given the fact that many RTC CCAs (e.g., GCC) have already been known for being too sensitive [1], further reducing the reaction would make the network layer overly sensitive to transient fluctuations, triggering unnecessary rate reductions that underestimate actual bandwidth and waste network capacity.

Second, current codecs inherently lag in adapting to rate changes at the default settings. We test the reaction time for three mainstream codecs – H.264, VP8, and AV1 – when the target bitrate is reduced from 30 Mbps to 3 Mbps, using their default low-latency settings. Recall the results in Figure 2, they all require 5–9 frames to align with new target bitrates at 30fps, translating to a reaction latency of 210–270ms. Critically, during this lag, the encoder continues producing frames at the higher rate, which exceeds the network's reduced capacity. This mismatch exacerbates queue buildup, pushing cumulative latency beyond 300–600ms. With the network layer's reaction time already minimized, the encoder's slow adaptation emerges as the primary bottleneck driving tail latency spikes.

Choice: A More Adaptive Encoder. Building on the preceding discussion, we propose to leverage more network information and enhance encoder-internal adaptation to address slow reaction during bandwidth drops. The goal is to develop a more adaptive encoder(MAE) that mitigates the gap between the network and codec layers.

Currently, CCAs treat encoders as black boxes: they only send target bitrates and initial parameters at the start, then leave encoders to self-regulate. To resolve this, encoders need to obtain more real-time network information from CCAs. This information would help encoders identify whether a

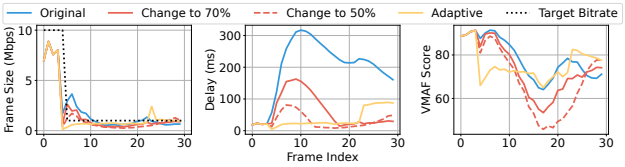


Figure 9: Directly reducing the target bitrate degrades quality when achieving competitive delay reduction compared to modifying buffer size.

bandwidth drop is occurring. The encoder could use this context to adaptively adjust its internal state, enabling rapid adaptation when a bandwidth drop happens.

3.2 Challenges

Nevertheless, it is non-trivial to make the encoder quickly follow bitrate drops without affecting the video quality, in a scalable manner for diverse network conditions.

How could MAE adapt to follow the bitrate drops? First, we need to determine which internal parameter in the encoder should be adapted. A straightforward idea is to directly scale down the target bitrate by a certain ratio. Yet, this approach is neither scalable nor efficient. Determining the appropriate ratio is non-trivial, as too aggressive a scaling would over-constrain the encoder, while too mild a scaling would fail to address overshoot, as shown in Figure 9. More critically, directly modifying the target bitrate would create an over-sensitivity issue—just like with reactive CCAs, at the cost of overall quality degradation.

To enforce strict adherence during bandwidth drops and preserve overall quality, MAE selects the encoder's virtual buffer size to regulate such adherence. Details are provided in §4.1.

How should MAE adapt to network changes? The next key challenge is how to balance the strict bitrate control and quality preservation. Current encoders already adjust the virtual buffer size to adapt to changes in the target bitrate, but this adjustment follows a fixed rule—the buffer size is always set to 0.5× the target bitrate, no matter the network's stability.

Our proposed adaptive approach retains this core mechanism but modifies the ratio used for this adjustment. In stable bandwidth conditions, we keep the conventional 0.5× ratio, which allows controlled overshoot and preserves video quality as intended. When a bandwidth drop is expected, we lower the ratio, which forces the encoder to adhere more strictly.

Additionally, we avoid the overuse of a low buffer ratio to avoid degrading overall video quality. As shown in Figure 10, the trade-off is clear: A permanently low ratio avoids excessive delay but causes persistent quality loss. The optimal approach, therefore, is to restrict the low ratio to only the short period of the bandwidth drop. This approach aims to reduce delay when it is most critical, while minimizing long-term quality loss throughout the entire stream.

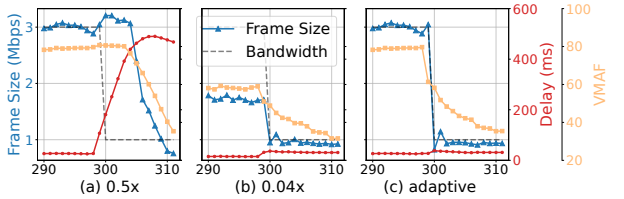


Figure 10: Frame sizes and delay with different vbv_buffer_size settings during the bandwidth drop from 3 Mbps to 1 Mbps. The x-axis represents the frame index. (a) set to half of the current bitrate; (b) set to the size of one frame; (c) adaptively chosen, using a smaller vbv_buffer_size during the drop while keeping the others at half of the current bitrate.

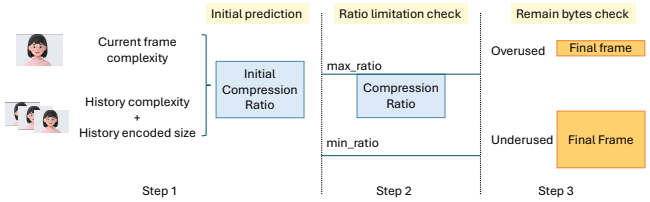


Figure 11: General encoder rate control process.

4 Design

4.1 Encoder Adaptation

We choose the virtual buffer size in the encoder's rate control module to adapt. The encoder's output size is shaped by multiple internal states and parameters, but virtual buffer size stands out as the most suitable for adaptive adjustment. To clarify this choice, we first contextualize the encoding workflow as shown in Figure 11: During encoding, the encoder first calculates a frame's compression ratio based on its content complexity (current and historical), used bit budget, and target bitrate. The rate control module then validates whether this compression ratio meets the bitrate requirement. In this process, there are several parameters influencing this final output.

Take the x264 encoder as an example: parameters tied to encoded size include max bitrate, compression ratio step limitation, and buffer size. However, the first two are unsuitable for adaptive adjustment due to distinct limitations: Max bitrate, when adjusted to respond to bandwidth drops, faces the same challenge as direct target bitrate modification, becoming either more sensitive or non-effective.

The compression ratio step limitation, which caps the maximum allowable difference in compression ratios between consecutive frames, also falls short, because its influence is confined to the "Ratio limitation check" phase. But in reality, during bandwidth drops, the bottleneck lies in the "Remain bytes check" phase. This means adjustments to the compression ratio step are overshadowed by rate control's stricter constraints, rendering them ineffective for timely adaptation.

By contrast, buffer size avoids these issues because it controls the core of bit allocation in the rate control process. Unlike max bitrate, which imposes a rigid, static upper bound, buffer size simply narrows the allowable range of short-term

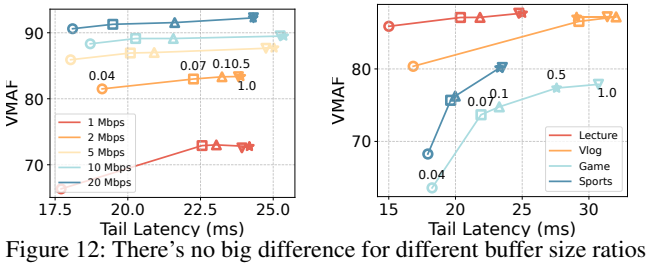


Figure 12: There's no big difference for different buffer size ratios when the network is stable.

bitrate fluctuations. This is critical: even with a smaller buffer, the encoder retains flexibility to allocate bits across frames based on content complexity. For example, a high-motion frame can still use more bits than a static one, as long as the total usage stays within the adjusted buffer bounds. This preserves the encoder's ability to prioritize quality for visually important frames, whereas rigid max bitrate limits would strip it away.

Encoder only adapts to network changes. We focus exclusively on network changes for three key reasons, grounded in both mechanism and experimental observation:

Network bandwidth drops are the dominant cause of tail latency spikes: Unlike content complexity fluctuations, bandwidth drops reduce the network's capacity to transmit data, leading to queued frames and cumulative delay. This is the primary source of tail latency in video streaming.

Optimal parameters for all network conditions or content types are unnecessary: Across different network states and content types, the conventional 0.5× buffer size already strikes a good balance. As shown in Figure 12, in stable conditions, delay is already low across all scenarios, and quality is preserved via controlled overshoots. For another, adapting the ratio to every scenario would expand the scope and duration of adjustments far beyond the narrow bandwidth drop window. This extended intervention would introduce more quality loss than needed, which contradicts our goal of balancing latency and quality.

Thus, our final strategy is: using a conventional 0.5× virtual buffer ratio under stable network conditions, switching to a lower ratio only when a bandwidth drop is detected.

4.2 Pre-reaction from Congestion Control

To enable timely encoder adaptation to bandwidth drops, we leverage the internal states of existing congestion control algorithms (CCAs) to pass signals about bandwidth drops. By lowering the threshold for triggering these signals, we let the CCA notify the encoder slightly earlier, before it formally confirms a bandwidth drop. This ensures the encoder can prepare for the impending bandwidth constraint.

We use CCA internal states to signal bandwidth drop.

Relying solely on target bitrate and its historical trends makes it hard to distinguish real bandwidth drops from transient fluctuations. Judging what counts as a "genuine" drop is non-trivial.

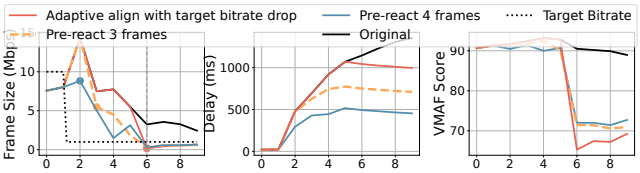
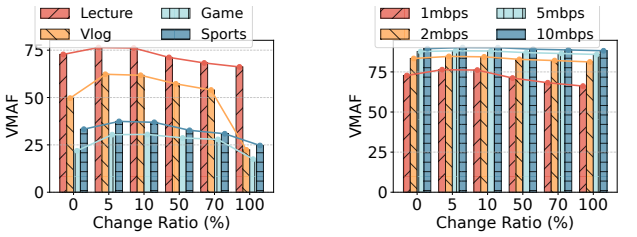
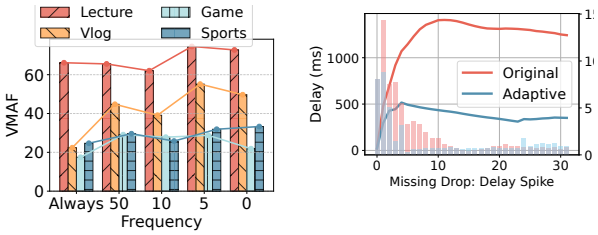


Figure 13: Pre-react adjustment of buffer size can mitigate latency spikes.



(a) Influence on Different Content. (b) Influence on Different Bitrate.



(c) Influence of Change Frequency. (d) Missing Drop: Delay Spike

Figure 14: Changing encoder buffer size won't influence quality significantly, but missing potential drop will cause a severe delay spike.

CCA internal states solve this by providing direct network context: Metrics like queue delays, jitter, or bottleneck estimates clarify whether a bitrate change stems from actual bandwidth constraints (not just encoder or content variations). This extra network-specific information enables more accurate identification of true drops.

Adopting pre-reaction for bandwidth drop signaling. To signal bandwidth drops to the encoder effectively, we adopt pre-reaction of the CCA. This choice is driven by two key considerations tied to real-time video's latency and Quality of Experience(QoE) requirements.

Latency accumulation is exponential with delayed response: As shown in Figure 13, for each frame of delayed adaptation after a drop, latency increases significantly, which is far harder to resolve than preventing them upfront. Furthermore, even if the CCA sensed bitrate has not decreased, the pre-reactive encoder can prevent overshooting a potential network drop, thereby significantly mitigating latency during an actual bandwidth reduction.

False positives are preferable to false negatives: A false positive (pre-reacting to a non-existent drop) may briefly tighten the encoder's buffer constraints, causing minor quality dips. However, a false negative (missing a real drop) allows unchecked frame overshoots, leading to cumulative delay,

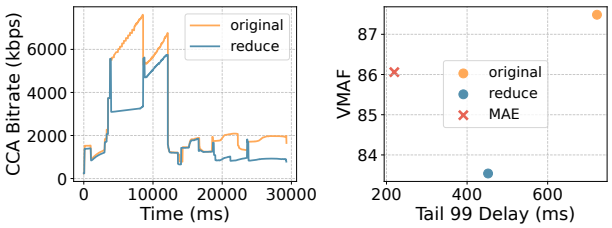


Figure 15: Directly reducing the CCA threshold degrades bandwidth utilization.

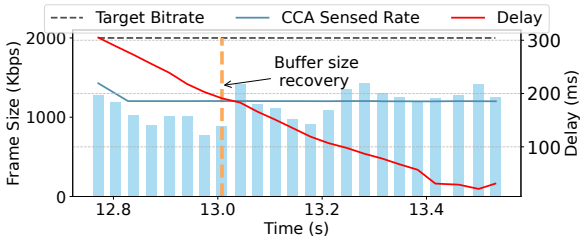


Figure 16: Based on CCA signal, the recovery is timely.

frozen frames, or packet drops—all of which severely harm QoE (Quality of Experience) in ways users find immediately noticeable. Figure 14(a) and 14(b) demonstrate that modifying the encoding buffer size for less than 10% of frames does not impact overall quality, regardless of the content type or bitrate level. Also, Figure 14(c) indicates that, for the same overall change duration, an increased frequency of changes leads to a reduction in quality. Fortunately, in real network scenarios, the drops detected by the CCA are infrequent, which minimizes the impact on quality. In contrast, Figure 14(d) illustrates that failing to detect potential drops can result in a latency increase of over 50%. Consequently, we prefer to manage controlled false positives to mitigate the substantial effects caused by false negatives.

Why can CCAs support pre-reaction? CCAs are well-suited to enable pre-reaction to bandwidth drops because of two key attributes rooted in the nature of network drops and the design of CCAs themselves.

First, bandwidth drops that cause latency spikes are inherently continuous rather than transient. To avoid mistaking fleeting fluctuations for real drops, CCAs typically rely on sliding time windows to confirm a drop. This means by the time a CCA formally decides to adapt to a drop, the bandwidth reduction has already been ongoing for a period. Given the continuous nature of such drops, this timing lag creates an opportunity for pre-reaction: Since the drop does not vanish abruptly, this earlier response remains valid, making pre-reaction theoretically sound.

Second, CCA design inherently incorporates delayed responsiveness to avoid overreactions. As noted previously, CCAs typically use sliding time windows to aggregate network data. Only when the aggregated trend in the window exceeds the threshold will the CCA confirm a drop. This deliberate delay creates an opportunity for pre-reaction: by either shrinking the window size or lowering the threshold, we can trigger a signal to the encoder earlier in the trend, before the CCA's standard logic would formally confirm the drop.

Why do we not directly tune CCA thresholds to enable pre-reaction? Adjusting CCA thresholds to trigger earlier rate reductions would harm network utilization. As shown in Figure 15, the unnecessary target bitrate drop at the 5th second will cause overall quality degradation. In contrast, our approach keeps CCA rate decisions intact but lets the encoder *prepare* for impending drops by tightening buffer constraints. This separation ensures the CCA maintains high utilization, while the encoder avoids overshoots that would force the CCA into drastic rate cuts.

When should we recover from the drop state? Recovery (reverting the encoder's buffer ratio to its original setting) is guided by CCA state transitions, ensuring timeliness without premature relaxation. Here's the mechanism:

When the CCA first detects potential congestion, it enters an "overuse" state and signals the encoder to tighten buffer constraints. This reduces the encoder's frame size overshoots, lowering the sending rate and allowing network queues to drain. As queues empty, RTT (Round-Trip Time) decreases—a change the CCA detects, transitioning to an "underuse" state (indicating the sending rate is now below available bandwidth). This "underuse" signal triggers the encoder to revert to its original buffer ratio, restoring quality flexibility.

As illustrated in Figure 16, when recovery is triggered by CCA signals, the CCA-sensed rate remains stable, and end-to-end latency still exceeds 200 ms. However, due to the CCA's inherent bandwidth underestimation during congestion alleviation, the current frame size has already aligned with the network's actual capacity, which eliminates the risk of further latency accumulation. This makes CCA-signal-driven recovery a timely and straightforward approach: it leverages the CCA's mature network state awareness to avoid premature or delayed adjustments. After recovery, the encoder can reactivate its bitrate overshoot capability, allowing occasional frame bitrate exceedances that enhance video quality.

The pre-reaction strategy is applicable to other mainstream CCAs. This signal and pre-reaction strategy is generalizable to other mainstream CCAs, as they share a core design trait: CCAs inherently use metrics and decision thresholds (or time windows) to avoid overreacting to noise. By tuning these thresholds or windows, we can extract earlier signals for pre-reaction, making the strategy adaptable across diverse congestion control algorithms. For example, Google Congestion Control(GCC) uses a 100-200ms time window to detect the bandwidth condition, COPA also employs queueing delay and minimal delay in a specific past time period. These give the potential to pre-trigger the signal for the encoder to react. Details are introduced in §5.2

5 Implementation

We implement our MAE within WebRTC (version m119), an open-source framework widely used for real-time audio and video communication, which integrates key components like congestion control, media encoding, and transport.

5.1 Encoder side implementation.

x264 implementation By default, we utilize x264 (Version 0.164.3191M) as our encoder, which is a popular open-source H.264 software encoder. Although native WebRTC uses OpenH264 for H.264 encoding because of licensing issues, we chose x264 for its superior rate control algorithms.

For real-time encoding, we utilize the recommended rate control approach, which is Average Bit Rate(ABR) paired with Variable Bitrate Video (VBV) buffer control. The VBV functions as a theoretical decoding buffer, regulating the remaining available bytes for encoding.

We select the VBV buffer size as our adaptive parameter to control the encoder's fluctuation and sensitivity to bitrate changes. Under normal network conditions, the VBV buffer size is set to half of the current bitrate, which gives half fps frame sizes flexibility. (e.g., 15 frames for 30 fps). When the CCA detects a potential drop signal, we reduce it to one frame size of the target bitrate, which can ensure that no frame overshoots the sensed target bitrate. Specifically, during bandwidth drops, the VBV buffer size is calculated using the following formula:

$$VBVBufferSize = \frac{TargetBitrate}{fps} + 1$$

$\frac{TargetBitrate}{fps}$ ensures the buffer size does not exceed one frame size, and the $+1$ offset avoids floating-point rounding errors.

Notably, we revise the x264 encoder as well to implement MAE. The modification still follows the standard so that the video stream is still compatible with any decoder.

5.2 Network side implementation.

We test the performance of MAE with two CCAs, GCC [7] and Copa [3]. Below we introduce how MAE probes the internal states from these two CCAs.

GCC implementation WebRTC's native GCC (Google Congestion Control) incorporates both delay-based and loss-based bandwidth estimators (BWE) to manage network congestion. For our adaptive strategy, we focus on the delay-based BWE, as it is the most responsive component for early detection of incipient bandwidth constraints.

The core of GCC's delay-based BWE is its trendline estimator, which analyzes the gradient of one-way delay variations over a sliding time window. In MAE, we set the sliding time window to 100 ms to achieve a timely reaction to bandwidth drops. By using this time window, we ignore transient bandwidth fluctuations within it. This estimator calculates a "slope" metric that reflects the rate at which queuing delays are increasing—an early indicator of network congestion. GCC then uses a predefined threshold to classify network states: if the slope exceeds this threshold, the algorithm transitions to an "overuse" state, signaling that the current sending rate exceeds network capacity. By using another slightly lower threshold, we can trigger another internal pre-reactive signal for the encoder to achieve our pre-reaction. Critically, this modification does not affect GCC's original rate control process; it only lets the encoder limit its overshooting capability.

Copa implementation Copa utilizes queueing delay to estimate network conditions. It sets the target rate to $\frac{1}{\delta d_q}$ packets per second, where $\delta = 0.5$ by default and $1/\delta$ is in units of MTU-sized packets. Then, it adjusts the congestion window based on its estimation of the sending rate and target rate. Copa already responds to network drops quickly by deciding the target rate based on the queueing delay. However, by setting a lower threshold on queueing delay to signal the encoder to reduce its virtual buffer size when this signal is triggered, we can further reduce the spike in queueing delay with a negligible drop in quality (VMAF).

6 Evaluation

This section evaluates the performance of our proposed MAE to validate its effectiveness, robustness, and generality. We first detail the experimental setup (Section 6.1) to ensure reproducibility, then address the following core research questions via trace-driven and real-world experiments:

- **Q1 Can the MAE reduce tail latency under real-world traces?** Compared to the status-quo solutions, MAE can reduce 86.2% stall rate and 54.7% P99 tail latency compared to the original WebRTC+x264 version.
- **Q2 Does the method reduce the video quality?** No, while reducing the tail latency, the adaptive encoder can maintain non-perceptual quality loss, keep the quality loss within 2.1% as the VMAF score is within 2.
- **Q3 Does the performance of MAE scale well with CCAs or network scenarios?** We test the performance of MAE over two different RTC CCAs (GCC and Copa), and three types of access network traces (WiFi, cellular, and Ethernet). Results consistently demonstrate the benefits of MAE.
- **Q4 What is the overhead of the proposed methods?** We further test the potential overhead in the video encoding by MAE. Results show that MAE is mostly equal to the original libx264 CPU and memory consumption, and also consistent encoding time.
- **Q5 How does Adaptive Encoder behave in real-world situations?** We further test the performance of MAE in the real world, MAE outperforms WebRTC+libx264 by 15.3% in stall number, with a quality loss of less than 2%.

6.1 Experiment Setup

Testbed. We have developed a testbed to automate the execution of experiments and analyze the results. The experiments are carried out on a single server, where the configuration is

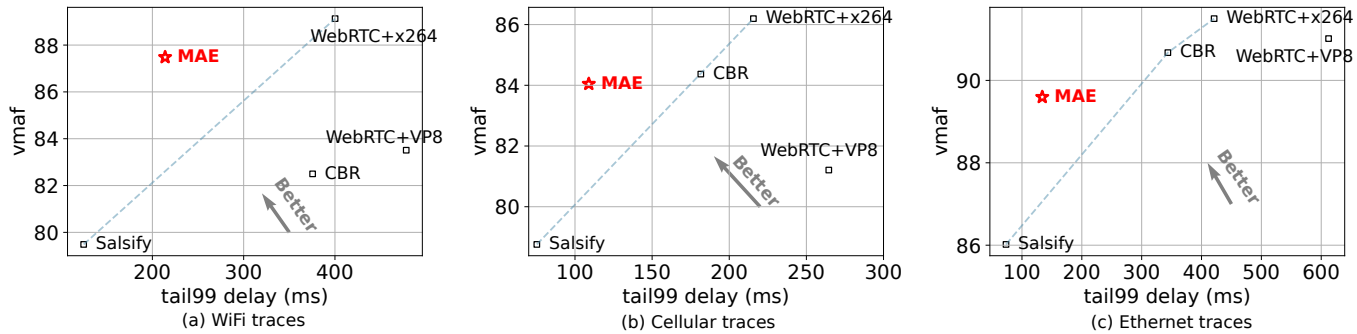


Figure 17: Comparison of average VMAF scores and stall numbers or tail99 delay of our design and baselines.

Intel(R) Xeon(R) Gold 6430 CPU (32 cores, 64 threads), 503 GB DDR5, Ubuntu 22.04.5 LTS. The downlink conditions of the receiver are managed using Mahimahi [31], a network emulator that replays actual network traces to simulate variations in bandwidth. Throughout the experiments, timestamps for frame transmission and reception (using the same time reference) are recorded and utilized to calculate end-to-end latency.

Video Dataset. We randomly selected 4 categories of videos from *YouTube UGC Dataset* [41], including Lecture, Music, Game and Vlog. The resolution is 1080p and the fps is fixed to 30. The overall duration of the video is more than 200 seconds.

Traces. We evaluate our method using real world network traces collected from diverse access networks (Wi-Fi, Cellular, and Ethernet) to validate that MAE generalizes across environments. These traces are sourced from Hairpin dataset [29], covering common real-world usage scenarios (e.g., mobile commuting, home Wi-Fi, wired office networks).

Metrics. In the evaluation, we focus on two core metrics to balance technical performance and user-centric quality:

(1) Tail Latency: We measure the 99th percentiles (P99) of end-to-end latency. These percentiles are standard for quantifying rare but impactful latency spikes—critical for real-time video.

(2) Stall Number: We adopt the definition of stalls from GSO and AnchorNet [24, 27], which categorizes latencies exceeding 200 ms as stalls. By utilizing this stall number metric, we aim to more accurately reflect its impact on user experience.

(3) Video Quality: We use Video Multimethod Assessment Fusion [34] (VMAF) as the primary metric. Unlike traditional metrics (e.g., PSNR, SSIM), VMAF incorporates human visual system models to align with subjective perception for the video instead of individual images, making it more reliable for evaluating noticeable quality changes to end users.

Baselines. We compare our Adaptive Encoder against 4 state-of-the-art baselines, ensuring fairness by aligning resolution, framerate, and initial bitrate across all:

- WebRTC-VP8: WebRTC's default VP8 encoder, the status-quo of the industry.
- WebRTC-X264: WebRTC with libx264 codec. We adopt the implementation from [21]
- CBR (WebRTC + x264 Constant Bitrate). We force the generated frames to follow the target size by turning on the feature of CBR.
- Salsify [16]: A low-latency streaming system optimized for real-time video. We implement Salsify following the setup used in Grace [8], and integrate it into our WebRTC-based testbed for a fair comparison.

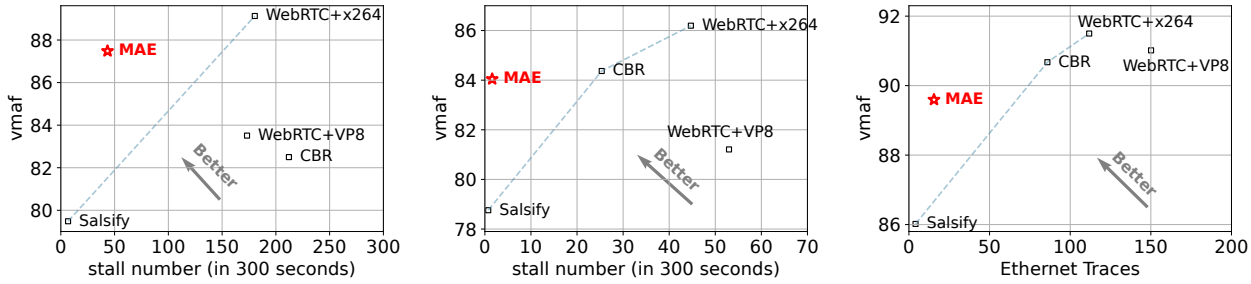
Unless otherwise specified, we use GCC for these baselines.

6.2 Overall Performance

We replayed 1320 seconds of real-world network traces (covering Wi-Fi, Cellular, and Ethernet) to confirm the generalization capability of MAE. Figure 17 presents key performance trade-off curves, plotting the 99th percentile latency against average VMAF scores across the three trace sets. Going to the upper left indicates higher quality paired with lower latency in the trade-off.

As shown in Figure 17, WebRTC+x264 delivers the highest quality but also exhibits the highest latency among all baselines. This is due to its slow adaptation to bandwidth drops: while it can leverage more bandwidth to preserve quality, this comes at the cost of large latency spikes. In contrast, Salsify exhibits the lowest latency but also the lowest VMAF scores. This is because Salsify basically ensures nearly every frame stays below the CCA-sensed bitrate, making it highly sensitive to bandwidth drops and thus reducing latency. However, this strict constraint underutilizes available bandwidth, limiting opportunities to enhance overall quality—resulting in VMAF scores 10% lower than those of the original WebRTC+x264. MAE, however, breaks this inherent trade-off by adaptively adjusting the encoder's overshoot capability based on real-time network conditions. As a result, MAE achieves a substantial 54.7% P99 tail latency reduction compared to the original WebRTC+x264, while keeping quality loss within 2.1%.

MAE outperforms all baselines across various network traces and video contents on tail latency and stalls. Across



(a) Wi-Fi traces (b) Cellular traces

Figure 18: Stall numbers vs VMAF quality over three network types.

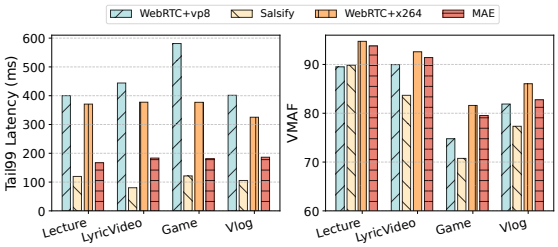


Figure 19: Comparison over video categories.

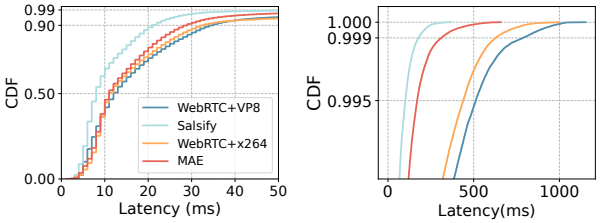


Figure 20: Latency distribution of all video frames.

different network environments, MAE demonstrates consistent superiority: it achieves 47-68% reductions in P99 latency for Wi-Fi, Cellular, and Ethernet traces, respectively. This consistent performance gain across diverse network types confirms MAE's robustness to varying network conditions.

Figure 18 further compares stall counts and VMAF scores across the three network categories. For stall reduction, MAE outperforms baselines by 75.9% (Wi-Fi), 86.1% (Cellular), and 86.1% (Ethernet). Stall counts were measured per trial, with each trial comprising 8,800 total frames. Collectively, these results translate to a significant drop in stall ratio—from the baseline's 3.8% to MAE's 0.69%.

Results over different videos. Figure 19 presents the performance of MAE and baselines across four distinct video types: low-motion content (LyricVideo), medium-motion content Lecture and Vlog), and high-motion content (Game). For all video categories, MAE achieves the lowest P99 latency while maintaining comparable VMAF quality to baselines.

When compared to Salsify—one of the lowest-latency baselines—MAE's stall rate is slightly higher (by 0.18% on average) but delivers a significant VMAF improvement of 6.5% overall. This quality gain is most pronounced in high-motion game videos, where MAE's VMAF score is 11% higher than Salsify's. Game content, which contains dense motion information, is particularly vulnerable to bandwidth under-utilization: Salsify's strict frame size constraints waste available bandwidth, while MAE's adaptive overshoot control preserves bandwidth for high-complexity frames, mitigating quality loss in dynamic scenes.

6.3 Other QoS Metric

We further analyze five additional QoS metrics to compare MAE with the baselines.

Latency Distribution. We present the Cumulative Distribution Function (CDF) of latency in Figure 20 to better understand how MAE controls the bufferbloat. Results show that at the general (i.e., average) latency level, MAE outperforms WebRTC+VP8 and WebRTC+x264. For the stricter P99.5 and P99.9 latency levels, MAE's maximum latency is capped at approximately 600 ms, while the maximum latency of WebRTC+VP8 and WebRTC+x264 exceeds 1000 ms.

Frame Rate. We further investigate the impact of dropping frames across different baselines. We measure the frame rate here and present the results on frame loss rate in Appendix A.1. As a result, the average frame rates across all experiments for Salsify, CBR, WebRTC+x264, and MAE are all close to 30 FPS, matching the original video's target frame rate. In contrast, WebRTC+VP8 exhibits a lower average frame rate of 27.7 FPS, which is attributed to VP8's internal frame dropping.

6.4 Ablation Study

We want to understand the effectiveness of each design choice in MAE. We decompose our method into two core components to quantify their individual contributions:

- *Adaptive VBV Buffer Only*: No pre-reaction mechanism; adjusts the virtual buffer size solely based on GCC's native overuse signals.
- *Full-function MAE*: Adaptively modifies the virtual buffer size using pre-reaction signals.

We present the results over the Wi-Fi traces in Figure 21. The results over cellular or Ethernet are similar. The "Adaptive

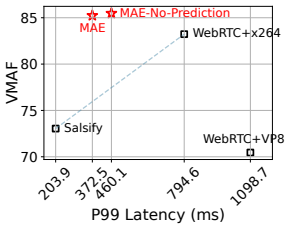


Figure 21: Ablation study.

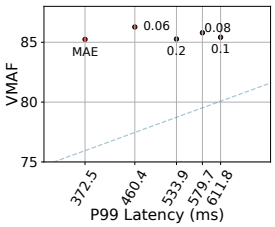


Figure 22: Sensitivity of buffer size.

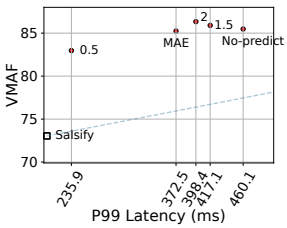


Figure 23: Sensitivity of pre-react threshold.

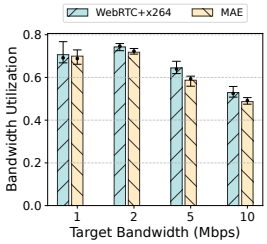


Figure 24: Average used bandwidth / Real network bandwidth.

VBV Buffer Only" variant (MAE-No-Prediction) shows that even without pre-reaction, merely adjusting the virtual buffer size can still reduce latency from 795 ms to 460 ms. The full MAE further reduces latency by 19% while maintaining comparable VMAF quality, confirming that both the adaptive buffer adjustment and pre-reaction mechanisms contribute effectively to performance improvements.

6.5 Parameter Sensitivity

We tested how key parameters influence performance to validate MAE's robustness against non-optimal settings, with results as follows:

First, we evaluated the adaptive buffer ratio during bandwidth drops—MAE uses a ratio of 0.04×. As shown in Figure 22, any reduction in the buffer size during bandwidth drops can reduce latency to a certain extent. Since this adjustment only affects a small subset of frames, it has minimal impact on overall quality (resulting in similar quality across different ratio settings). Among the tested ratios, setting the buffer size to 0.04× achieves a significant latency reduction during drops, making it outperform other parameter values.

Second, we analyzed the sensitivity of the pre-reaction mechanism. For GCC, we first set the overuse threshold to half of its original value. We later test the scaling factor over the threshold that the current network trend must exceed to trigger buffer adjustment. For example, if the scaling factor is 0.5, buffer adjustment is triggered when the network trend reaches 50% of the threshold (higher sensitivity). As shown in Figure 23, higher sensitivity reduces latency but leads to a 2.3% quality drop. Conversely, lower sensitivity yields a slight quality improvement but increases latency. MAE selects 1 as the optimal trade-off value to balance latency control and quality preservation.

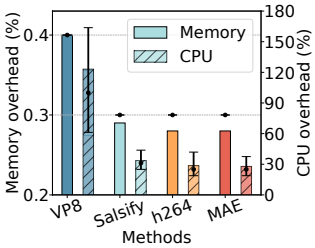


Figure 25: CPU usage comparison in real network traces.

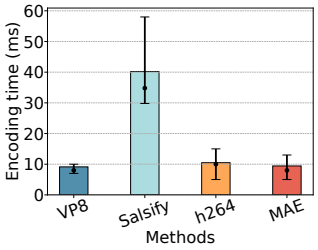


Figure 26: Encoding latency.

6.6 Microbenchmarks

Link Utilization. We further want to investigate the influence of MAE on link utilization and the potential interaction with CCAs. We evaluate MAE's link utilization against the baseline WebRTC+x264, with results shown in Figure 24. Because MAE only reduces the virtual buffer size for a small portion of frames, its overall bandwidth utilization is nearly identical to WebRTC+x264 across different target bandwidth levels, especially at low target bandwidths. For higher target bandwidths (e.g., 10 Mbps), at most 4% reduction in utilization barely impacts video quality, while reserving bandwidth headroom to mitigate potential bandwidth drops.

Encoding Time. Figure 26 presents the distribution of encoding latency for various baselines versus MAE. On average, MAE's encoding time is slightly longer than VP8's (by 0.3 milliseconds) but shorter than Salsify's. Notably, Salsify uses a custom-designed encoder, which incurs significantly longer encoding time compared to commercial encoders (e.g., VP8, x264). Comparison with x264 further shows that our adaptive adjustments do not impact encoding time—MAE's encoding latency remains on par with WebRTC+x264.

Runtime Overhead. We evaluated the overall system overhead of MAE, with results shown in Figure 25. MAE maintains CPU and memory usage at the same level as Salsify and WebRTC+x264, and significantly lower than VP8. This is because VP8 has lower encoding efficiency, which consumes more computational resources. Among the three methods (Salsify, MAE, WebRTC+x264), Salsify's CPU usage is slightly higher than MAE's, while MAE's is marginally higher than WebRTC+x264's. Corresponding to this trend, their encoding complexity ratio follows Salsify > MAE > WebRTC+x264, confirming that higher encoding complexity correlates with increased CPU consumption.

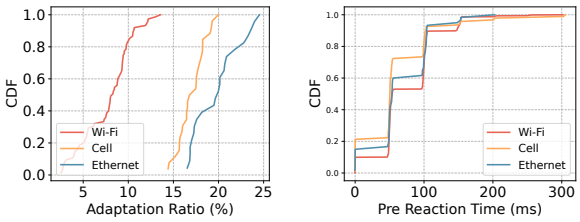
Scalability to Other CCAs. To validate MAE's generality, we extended our method to the Copa CCA and tested it on the Wi-Fi traces. As shown in Table 1, MAE still achieves a 47.5% reduction in the stall ratio while maintaining VMAF quality at the same level as the original libx264. This demonstrates that our method can be extended to other CCA algorithms and still deliver significant performance benefits.

6.7 MAE Deep Dive

To better understand MAE's performance, we conduct an in-depth analysis of the experimental results:

Table 1: Evaluation results over the Copa CCA.

Method	Stall Ratio	VMAF
WebRTC+libx264	0.19	83.12
WebRTC+MAE	0.10	88.36



(a) Adaptation ratio. (b) Pre-reaction time.

Figure 27: The statistical distribution when running MAE.

Adaptation Ratio. As noted in §4.1, MAE is designed to only reduce the VBV buffer for a small portion of time. We therefore analyzed the adaptation percentage (defined as the share of adapted frames relative to total frames) for each trial. Figure 27(a) presents the CDF of this adaptation ratio across different network types. Results show that for Wi-Fi traces, 80% of trials have an adaptation ratio below 10%—the reason why there is no noticeable quality loss. On the other hand, this also demonstrates that the impact of false positive of bitrate drops is limited. For Cellular and Ethernet traces, similarly, the adaptation ratio maintains around 20% on average. This difference arises because the Cellular and Ethernet traces used in our tests are more fluctuating than the Wi-Fi traces, leading to more frequent encoder adaptations.

Pre-reaction Time. We utilize a reduced threshold to enable pre-reactions, but how does this mechanism perform in practice? We measured the pre-reaction time across all real-world traces to quantify its effectiveness. We define a "real network drop" as an event where the trend of five consecutive frames exceeds the threshold. Based on this definition, we analyzed how often pre-reactions successfully anticipate real drops. Figure 27(b) shows that across all network types, over 20% of drops are pre-reacted to more than 100ms in advance. This pre-reaction window creates valuable buffer headroom to mitigate the impact of impending bandwidth drops.

6.8 Real-World Experiments

We further conduct real-world experiments to evaluate the user-centric QoE of MAE. We set up an RTC server and a client with WebRTC, and they are connected over Wi-Fi. The videos from 4 categories (Lecture, Music, Game, and Vlog) in *YouTube UGC Dataset* [41] are concatenated into a video clip and streamed from the server to the client in real time. To ensure fairness, we rotate through different encoders, including MAE and the four baselines, each for ten minutes. The total testing time is 6 hours.

We measure the frame latency by calculating the timestamp difference between the video sent and the video received. The VMAF is also evaluated by comparing the received video and the original video. As shown in Table 2, MAE outperforms WebRTC+libx264 by 15.3% in stall number, with a quality loss of less than 2%.

Table 2: Performance of MAE in real-world experiment.

Method	Stall Number	VMAF
WebRTC+vp8	131	96.44
Salsify	7	86.92
WebRTC+libx264	65	97.46
MAE	**55**	**95.28**

The results indicate that MAE provides enhanced QoE in real-world environments.

7 Discussion

Scenario Limitation. This work limits encoder adaptation ratio to a small portion, ignoring content complexity fluctuations. Yet content variations (e.g., high-motion frames) can also cause overshoots. Future work may integrate content prediction to distinguish network- vs. content-induced overshoots.

Encoder Generalization. Validated on x264, the MAE approach can extend to VP8, AV1, etc. These encoders share similar rate control mechanisms (e.g., token buckets, rate windows). Its adaptive buffer and pre-reaction logic can be applied to most modern RTC encoders. We leave these implementations as our future work.

8 Conclusion

In this paper, we identify an issue that the bufferbloat shifts to the video encoder in the delivery pipeline. We later introduce MAE, which is a novel adaptive framework that optimizes video transmission latency and quality by combining dynamic virtual buffer size adjustment with pre-reaction to network bandwidth trends. Experiments across Wi-Fi, Cellular, and Ethernet traces, plus diverse video types, show MAE outperforms baselines consistently: it cuts stall rate by 86.2% and P99 latency by 54.7%, while keeping VMAF quality loss within 2.1%, with minimal CPU/memory overhead and strong scalability to other CCAs and encoders. MAE aims to provide insights into an often-ignored part of the entire RTC pipeline—encoder—arguing that the content source also needs to be more adaptive to network bandwidth.

Acknowledgements. We thank our shepherd, Dave Oran, and the anonymous NSDI reviewers for valuable comments. This work is supported by RGC Early Career Scheme (26212525) and Guangdong Basic and Applied Basic Research Fund (2025A1515010460). Zili Meng is the corresponding author.

References

[1] Agarwal, N., Pan, R., Yan, F. Y., and Netravali, R. Mowgli: Passively learned rate control for {Real-Time} video. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)* (2025), pp. 579–594.

[2] ANAND, A., MUTHUKRISHNAN, C., KAPPES, S., AKELLA, A., AND NATH, S. Cheap and large cams for high performance data-intensive networked systems. In *NSDI* (2010), vol. 10, pp. 29–29.

[3] ARUN, V., AND BALAKRISHNAN, H. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 329–342.

[4] BELAY, A., PREKAS, G., KLIMOVIC, A., GROSSMAN, S., KOZYRAKIS, C., AND BUGNION, E. {IX}: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 49–65.

[5] BERMUDEZ, H.-F., MARTINEZ-CARO, J.-M., SANCHEZ-IBORRA, R., ARCINIEGAS, J. L., AND CANO, M.-D. Live video-streaming evaluation using the itu-t p. 1203 qoe model in lte networks. *Computer Networks 165* (2019), 106967.

[6] BLUM, N., LACHAPELLE, S., AND ALVESTRAND, H. Webrtc: Real-time communication for the open web platform. *Communications of the ACM 64*, 8 (2021), 50–54.

[7] CARLUCCI, G., DE CICCO, L., HOLMER, S., AND MASCOLO, S. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems* (2016), pp. 1–12.

[8] CHENG, Y., ZHANG, Z., LI, H., ARAPIN, A., ZHANG, Y., ZHANG, Q., LIU, Y., DU, K., ZHANG, X., YAN, F. Y., ET AL. {GRACE}:{Loss-Resilient}{Real-Time} video through neural codecs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)* (2024), pp. 509–531.

[9] CHESHIRE, S. Source Buffer Management. Internet-Draft draft-cheshire-sbm-02, Internet Engineering Task Force, Mar. 2025. Work in Progress.

[10] CHUN, B.-G., DABEK, F., HAEBERLEN, A., SIT, E., WEATHERSPOON, H., KAASHOEK, M. F., KUBIATOWICZ, J., AND MORRIS, R. T. Efficient replica maintenance for distributed storage systems. In *NSDI* (2006), vol. 6, pp. 4–4.

[11] DABEK, F., LI, J., SIT, E., ROBERTSON, J., KAASHOEK, M. F., AND MORRIS, R. T. Designing a dht for low latency and high throughput. In *NSDI* (2004), vol. 4, pp. 85–98.

[12] DASARI, M., KAHATAPITIYA, K., DAS, S. R., BALASUBRAMANIAN, A., AND SAMARAS, D. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 103–118.

[13] FLACH, T., DUKKIPATI, N., TERZIS, A., RAGHAVAN, B., CARDWELL, N., CHENG, Y., JAIN, A., HAO, S., KATZ-BASSETT, E., AND GOVINDAN, R. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (2013), pp. 159–170.

[14] FLAVEL, A., MANI, P., MALTZ, D., HOLT, N., LIU, J., CHEN, Y., AND SURMACHEV, O. {FastRoute}: A scalable {Load-Aware} anycast routing architecture for modern {CDNs}. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), pp. 381–394.

[15] FLORES, M., WENZEL, A., AND KUZMANOVIC, A. Enabling router-assisted congestion control on the internet. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)* (2016), IEEE, pp. 1–10.

[16] FOULADI, S., EMMONS, J., ORBAY, E., WU, C., WAHBY, R. S., AND WINSTEIN, K. 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)* (2018), pp. 267–282.

[17] GHADIYARAM, D., PAN, J., AND BOVIK, A. C. A subjective and objective study of stalling events in mobile streaming videos. *IEEE Transactions on Circuits and Systems for Video Technology 29*, 1 (2017), 183–197.

[18] GOYAL, P., AGARWAL, A., NETRAVALI, R., ALIZADEH, M., AND BALAKRISHNAN, H. {ABC}: A simple explicit congestion controller for wireless networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 353–372.

[19] GROSVENOR, M. P., SCHWARZKOPF, M., GOG, I., WATSON, R. N., MOORE, A. W., HAND, S., AND CROWCROFT, J. Queues {don't} matter when you can {JUMP} them! In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), pp. 1–14.

[20] HU, J., ZHOU, Z., AND YANG, X. Characterizing {Physical-Layer} transmission errors in cable broadband networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 845–859.

[21] HUANG, X., XU, J., WANG, H., YU, H., SATHYANARAYANA, S. D., SHI, S., AND MENG, Z. Ace: Sending burstiness control for high-quality real-time communication. In *To appear at Proceedings of the ACM SIGCOMM 2025 Conference* (2025).

[22] LI, T., ZHENG, K., XU, K., JADHAV, R. A., XIONG, T., WINSTEIN, K., AND TAN, K. Tack: Improving wireless transport performance by taming acknowledgments. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 15–30.

[23] LI, Y., QIU, J., WANG, H., LI, Z., QIAN, F., YANG, J., LIN, H., LIU, Y., XIAO, B., QIN, X., ET AL. Dissecting and streamlining the interactive loop of mobile cloud gaming. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)* (2025), pp. 595–611.

[24] LIN, X., MA, Y., ZHANG, J., CUI, Y., LI, J., BAI, S., ZHANG, Z., CAI, D., LIU, H. H., AND ZHANG, M. Gso-simulcast: global stream orchestration in simulcast video conferencing systems. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 826–839.

[25] LLOYD, W., FREEDMAN, M. J., KAMINSKY, M., AND ANDERSEN, D. G. Stronger semantics for {Low-Latency}{Geo-Replicated} storage. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (2013), pp. 313–328.

[26] MARTINEZ-CARO, J.-M., AND CANO, M.-D. On the identification and prediction of stalling events to improve qoe in video streaming. *Electronics 10*, 6 (2021), 753.

[27] MENG, T., ZHANG, W., CHEN, D., WANG, Z., LI, Q., YAN, C., YANG, W., YUAN, C., ZHANG, L., KUANG, J., ET AL. {AnchorNet}: Bridging live and collaborative streaming with a unified architecture. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)* (2025), pp. 1021–1036.

[28] MENG, Z., GUO, Y., SUN, C., WANG, B., SHERRY, J., LIU, H. H., AND XU, M. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 193–206.

[29] MENG, Z., KONG, X., CHEN, J., WANG, B., XU, M., HAN, R., LIU, H., ARUN, V., HU, H., AND WEI, X. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)* (2024), pp. 907–926.

[30] MENG, Z., WANG, T., SHEN, Y., WANG, B., XU, M., HAN, R., LIU, H., ARUN, V., HU, H., AND WEI, X. Enabling high quality {Real-Time} communications with adaptive {Frame-Rate}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1429–1450.

[31] NETRAVALI, R., SIVARAMAN, A., DAS, S., GOYAL, A., WINSTEIN, K., MICKENS, J., AND BALAKRISHNAN, H. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (2015), pp. 417–429.

[32] NI, Y., ZHENG, Z., LIN, X., GAO, F., ZENG, X., LIU, Y., XU, T., WANG, H., ZHANG, Z., DU, S., ET AL. Cellfusion: Multipath vehicle-to-cloud video streaming with network coding in the wild. In *Proceedings of the ACM SIGCOMM 2023 Conference* (2023), pp. 668–683.

[33] OUSTERHOUT, A., FRIED, J., BEHRENS, J., BELAY, A., AND BALAKRISHNAN, H. Shenango: Achieving high {CPU} efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (2019), pp. 361–378.

[34] RASSOOL, R. Vmaf reproducibility: Validating a perceptual practical video quality metric. In *2017 IEEE international symposium on broadband multimedia systems and broadcasting (BMSB)* (2017), IEEE, pp. 1–2.

[35] RICHARDSON, I. E. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[36] SCHWARZ, H., MARPE, D., AND WIEGAND, T. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology 17*, 9 (2007), 1103–1120.

[37] SEUFERT, M., CASAS, P., WEHNER, N., GANG, L., AND LI, K. Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2019), IEEE, pp. 688–695.

[38] TARAGHI, B., NGUYEN, M., AMIRPOUR, H., AND TIMMERER, C. Intense: In-depth studies on stall events and quality switches and their impact on the quality of experience in http adaptive streaming. *IEEE Access 9* (2021), 118087–118098.

[39] ULUYOL, M., HUANG, A., GOEL, A., CHOWDHURY, M., AND MADHYASTHA, H. V. {Near-Optimal} latency versus cost tradeoffs in {Geo-Distributed} storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 157–180.

[40] WANG, T., MENG, Z., XU, M., HAN, R., AND LIU, H. Enabling high frame-rate uhd real-time communication with frame-skipping. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges* (2021), pp. 19–24.

[41] WANG, Y., INGUVA, S., AND ADSUMILLI, B. Youtube ugc dataset for video compression research. In *2019 IEEE 21st international workshop on multimedia signal processing (MMSP)* (2019), IEEE, pp. 1–5.

[42] WU, Z., YU, C., AND MADHYASTHA, H. V. {CosTLO}:{Cost-Effective} redundancy for lower latency variance on cloud storage services. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), pp. 543–557.

[43] YASUKATA, K., HONDA, M., SANTRY, D., AND EGGERT, L. {StackMap}:{Low-Latency} networking with the {OS} stack and dedicated {NICs}. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (2016), pp. 43–56.

A Supplementary Experiments

A.1 Frame Skipping Rate

Enabling frame dropping would introduce significant bias in video quality assessment. To ensure consistency and fairness across all measurements, we disabled network-level frame dropping. As a result, WebRTC+x264, Salsify, CBR, and MAE all exhibit a 0% loss rate; the loss rate of WebRTC+VP8 is presented in Figure 28. This loss rate stems purely from VP8's intermittent frame dropping, as VP8's rate control mechanism automatically drops frames to match the target bitrate. However, as overall results show, such frame dropping degrades quality: despite being one of the most advanced encoders in our baselines, VP8's frame dropping causes its overall VMAF score to be lower than MAE's.

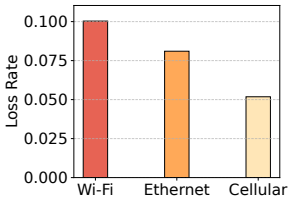


Figure 28: WebRTC+VP8 loss rate in different network type.