Tight Loops, Smooth Streams: Responsive Congestion Control for Real-Time Video

Pantea Karimi
Massachusetts Institute of Technology, USA

Sadjad Fouladi
Independent Researcher (work performed while at Microsoft Research), USA

Vibhaalakshmi Sivaraman
Amazon, USA

Mohammad Alizadeh
Massachusetts Institute of Technology, USA

Abstract

Real-time video streaming relies on rate control to match video bitrate to network capacity while keeping latency low. Existing deployed video rate controllers react slowly to network changes, causing under-utilization and latency spikes. In contrast, modern delay-sensitive congestion control algorithms (CCAs) adapt on round-trip-time timescales, maintaining a tight feedback loop that achieves both high utilization and low latency.

We introduce Vidaptive, a lightweight framework that enables real-time video to leverage responsive CCAs without codec changes. Vidaptive decouples encoding from transmission: it paces video frames at the CCAs rate and injects dummy packets when the encoder output is insufficient, preserving a continuous feedback loop. An online algorithm dynamically adjusts the encoders target bitrate to align with CCA capacity while bounding frame latency.

Implemented in Google WebRTC, Vidaptive improves both video quality and tail latency on diverse cellular traces. Compared to GCC, it delivers 1.5× higher bitrate, +40% VMAF, +1.4 dB SSIM, +1.3 dB PSNR, and reduces 95th-percentile frame latency by 57% (2.2 seconds). Against Salsify, it achieves lower tail latency without invasive codec modifications. These results show that coupling existing CCAs with a thin adaptation layer can outperform specialized video rate controllers while remaining deployable in practice.

2012 ACM Subject Classification Networks → Transport protocols

Keywords and phrases real-time video, congestion control, transport protocols, video rate control, low-latency video communication, tight feedback loop

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.9

Acknowledgements We thank Keith Winstein for countless conversations about this work over the years; this version of the paper would not have been possible without Keith's insights. We also thank Weiyang "Frank" Wang and Sudarsanan Rajasekaran for their valuable feedback and support. We are grateful to Behnaz Arzani and Siva Kesava Reddy Kakarla for their insightful comments, particularly on framing the narrative of the paper. Finally, we thank our shepherd, Zili Meng, for their guidance throughout the shepherding process.

1 Introduction

Real-time video has become ubiquitous on the Internet, enabling applications such as video conferencing (e.g., Teams, Google Meet), cloud gaming, and teleoperation [13, 40]. Unlike on-demand video streaming, real-time video faces two key challenges. First, latency is critical: in cloud gaming, for example, even small delays could make the game unplayable [46]. Second, unlike on-demand streaming, which pre-fetches and buffers content to mask network variability, real-time video must adapt instantaneously to network fluctuations.

A critical component in meeting latency targets for real-time video is the rate controller, which continuously adjusts the video bitrate to match available network capacity. Since buffering is virtually absentand fundamentally at odds with low-latency operationrate control becomes the main mechanism for balancing strict latency requirements with high video quality. State-of-the-art rate controllers for real-time video, however, have significant shortcomings: they respond slowly (up to seconds) to changing network conditions, causing both under-utilization and latency spikes. This problem is well recognized, and numerous works have examined it [21, 42, 46, 5, 35, 33].

Real-time video has two coupled objectives: maximizing delivered visual quality ($\propto$ goodput) while keeping frame latency low ($\propto$ packet latency). Many delay-sensitive congestion control algorithms (CCAs) are designed for the same goal [7, 10, 50, 55, 18]. These CCAs can adapt their sending rate to changes in available bandwidth within just a few round-trip times (10s of milliseconds). By contrast, in our experiments on cellular traces, Google Congestion Control (GCC) [11, 9]—the default production rate controller in Google's WebRTC stack[1]— used only one-third of the available bandwidth compared to Copa [7], while suffering 7$\times$ higher P95 packet latency. This problem is not unique to GCC; prior work has shown that other real-time video rate controllers like Skype also struggle to adapt to bandwidth variations quickly [50].

Why do video rate controllers perform poorly in variable networks compared to state-of-the-art CCAs? As shown in prior work [21], the fundamental issue is that the rate controller cannot precisely control the size of the video frames produced by the video encoder. Best-in-class CCAs [7, 10] are effective when the sender always has data ready to send — in other words, when the flow is *backlogged*. For a backlogged flow, the CCA can precisely control the timing of transmissions and the number of packets in flight. Having packets constantly in flight maintains a *tight feedback loop* between the sender and the network. Each packet acts as a probe and the controller can rapidly estimate the available bandwidth. Without packets on the wire, however, this loop breaks, and the sender is left blind to the networks current state. This is particularly problematic on networks where bandwidth is highly variable, such as wireless networks.

In real-time video, the video encoder emits frames at fixed intervals. The frame size depends on content complexity, video motion, codec settings, and the target bitrate. The encoders output is inherently unpredictable. The rate controller *requests* a target rate, but the encoder cannot deliver it precisely (see Fig. 1a). The resulting mismatch makes it difficult to track variable bandwidth effectively. Small frames underutilize the link and disrupt the CCA's feedback loop; oversized frames overshoot capacity, inducing queues and delay.

Since the encoder's output cannot be accurately controlled, most systems adopt a conservative strategy, prioritizing latency over video quality. The rate controller increases the video bitrate slowly and leaves significant headroom to avoid queue buildup—at the cost of lower utilization [11, 9]. This is the approach taken by WebRTC and GCC.

A second approach, proposed by Salsify [21], addresses the problem from the encoder side. It redesigns the encoder to generate multiple candidate encodings of each frame, enabling the transport layer to select the largest frame that fits within the available capacity. This architecture is compelling, but it requires intrusive codec modifications. Since production codecs are highly optimized systems (often with hardware acceleration), such a forklift

[1] WebRTC is the engine behind many browser-based real-time video applications. Since the publication of the original GCC paper, the algorithm deployed in WebRTC has evolved; we use version M108 [16] in our study.

change to the codec is difficult to deploy.

Do we need specialized transport protocols or new codecs? We set out to understand if it is possible to reuse best-in-class CCAs for real-time video, retaining their strong performance in throughput, delay, and responsiveness. We developed Vidaptive, a framework that introduces a thin layer between existing unmodified encoders and CCAs, enabling best-in-class CCAs to operate at full potential. Vidaptive significantly improves responsiveness to network variability and its design builds on two key ideas.

Our first idea is to decouple packet transmissions from the encoders output. Vidaptive treats video streams as backlogged flows for rate control: if the encoder produces more packets than the CCA can send, they are queued and paced at the CCAs rate; if it produces too few packets, Vidaptive injects dummy packets to satisfy CCA's sending rate. On the wire, this novel scheduling of dummy packets makes Vidaptive behave like a continuously backlogged CCA, keeping the feedback loop intact and yielding more accurate bandwidth estimates.

Our second idea is a set of techniques to align the video bitrate with the CCAs sending rate while keeping frame latency bounded. To guide the encoder, Vidaptive runs an online algorithm that sets the target bitrate based on the CCAs rate and recent frame transmission times. It measures frame transmission times and maintains a *headroom*the gap between the encoders bitrate and the CCAs rateneeded to meet a desired frame delay. By tracking the distribution of observed delays, Vidaptive adapts this headroom to both encoder and network variability. The amount of headroom provides a simple knob to trade off video quality and latency, enabling operating points that were not possible before. For example, video conferencing can tune Vidaptive for ultra-low latency, while live streaming can prioritize video quality at the cost of modestly higher latency.

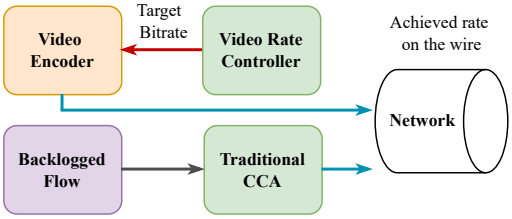
Vidaptive shows that we may not need specialized transport protocols for real-time video (e.g., Salsify [21], GCC [11], Pudica [46]); existing CCAs can suffice with the right architecture. The key aspect of this architecture is to enable the CCA to maintain a tight feedback loop for accurately tracking network bandwidth changes. Vidaptive achieves this by supplying extra traffic as needed to keep the flow backlogged. It may be counterintuitive to send extra packets in variable network conditions when we have strict latency targets. Yet surprisingly, our results show that, even with the overhead of extra traffic, video tail latency and quality both improve compared to existing rate controllers. The tight feedback loop of the CCA is the main driver of this improvement, and its advantage in estimating the available bandwidth accurately is so substantial that it outweighs the cost of extra traffic.

We implemented Vidaptive on top of WebRTC (M108 [16]). On a set of cellular traces, Vidaptive improves the video quality and tail latency. It increases average video bitrate by 1.5×, increases SSIM by 1.4 dB and Video Multi-Method Assessment Fusion (VMAF) [32] by 40%, while reducing P95 frame latency by 2.2 s, a 57% reduction. Against Salsify, Vidaptive achieves markedly lower tail latency while requiring no codec changes. We show that Vidaptive seamlessly adapts two CCAs: Copa and ROCC [4], a BBR variant, for real-time video.

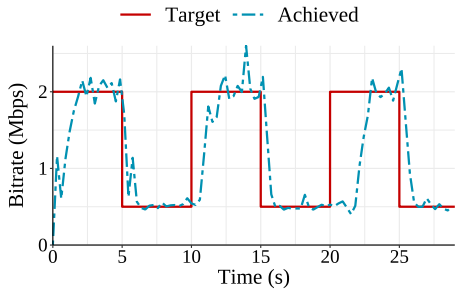
2 Motivation and Key Ideas

2.1 The Problem

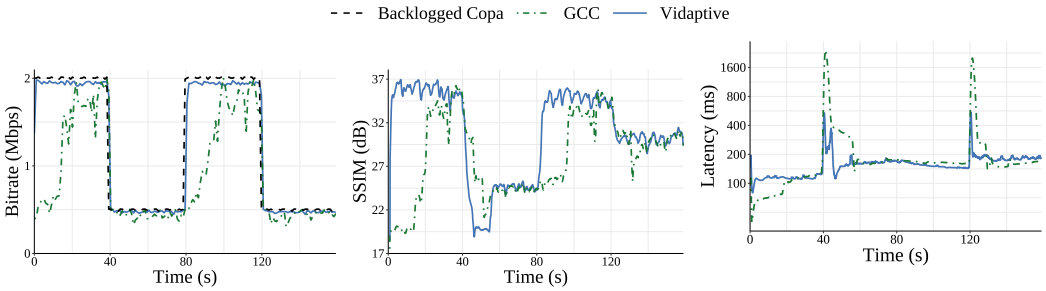
Status Quo for Video Rate Control. To see how todays production systems handle real-time video, we evaluate Google Congestion Control (GCC) [11], the rate control mechanism inside WebRTC, on a simple link. WebRTC is the engine behind browser-based real-time



(a) In current video rate controllers, unlike traditional CCAs, instantaneous rate on the wire is dictated by the video encoder.



(b) Video encoder has a sluggish "Achieved" response to a time-varying "Target" input bitrate.



(a) Link Utilization **(b)** Frame Quality **(c)** Frame Latency

Figure 2 Utilization, frame quality, and latencies of Copa on a backlogged flow, GCC on a video flow, and Vidaptive on a video flow. GCC is slow to match the available capacity and under-utilizes the link in the steady state. Vidaptive responds much faster to link variations and is similar to Copa's performance on a backlogged flow.

applications, including Google Meet and Microsoft Teams. We ran a recent version of WebRTC (M108 [16]) on an emulated link that alternates between 2 Mbps and 500 Kbps every 40 seconds. The minimum round-trip time (RTT) is 50 ms, and the bottleneck buffer is large enough to avoid packet drops.

As shown in Fig. 2a, GCC is sluggish when ramping up: at the start of the stream and again when the link recovers to 2 Mbps at $t = 80$s, it takes *18 seconds* to climb from 500 Kbps to 2 Mbps, resulting in lower visual quality during that time (Fig. 2b). GCC is also slow to react to link drops: when the link rate falls to 500 Kbps at $t = 40$s, frame latency spikes to over a second and takes 12 *seconds* to stabilize (Fig. 2c).

Contrast this behavior with traditional congestion control algorithms [7, 4, 8, 10] operating on *backlogged* flows that always have packets waiting to be sent. They respond to such network events much faster, typically over few RTTs. For instance, the "Backlogged Copa" lines in Fig. 2a shows that Copa [7] reacts quickly on the same time-varying link. This wide disparity raises the question: *Why does state-of-the-art video rate control lag so far behind state-of-the-art congestion control?*

Encoder-driven Rate Control. In video rate controllers like GCC, the *instantaneous rate* on the wire is dictated by the frame sizes produced by the encoder (see Fig. 1a), unlike traditional CCAs, where the controller itself regulates the rate on the wire. GCC controls the video bitrate by adapting the encoder's target rate, but the encoded frame sizes can be highly variable. The encoder achieves its target bitrate only on average–usually over several frames [21], and cannot immediately adapt to sudden changes. We illustrate this in Fig. 1b

where we supply the VP8 encoder with a target bitrate that switches between 2 Mbps and 500 Kbps every 5s. Every time the bitrate goes up from 500 Kbps to 2 Mbps, the encoder takes nearly 2 seconds to catch up. On the way down from 2 Mbps to 500 Kbps, it takes about a second to lower the bitrate. The reason for this lag is that the size of an encoded frame depends on factors such as quantization parameters, the encoder's internal state, and the motion in the video, and it is only known accurately after encoding [21]. The encoder tries to rectify over- and under-shootings by adjusting the quality of subsequent frames, thus, it shows significant per-frame variance around the average.

Because the encoder cannot match the target bitrate on a per-frame timescale, GCC cannot quickly reduce its rate when capacity drops. Instead, GCC has to be conservative and leave abundant *headroom* at all times. Even in the steady state, GCC limits utilization at about 85% and still experiences occasional latency spikes (Fig. 2c). Headroom lowers the risk of a latency spike during fluctuations but makes it difficult to track the available bandwidth accurately because it breaks the tight feedback loop from the network. For instance, GCC relies on delay gradients to estimate link rate, but when the link is underutilized, queues do not build and gradients provide no signal. As a result, GCC ramps up slowly to discover capacity. Recall that when GCC increases its target bitrate, the encoder takes a few seconds to catch up. GCC must wait for the video bitrate to increase before it knows if it is safe to increase it further, so the entire process takes 15–20 seconds to converge (Fig. 2a).

Basic probing mechanisms are not enough. Video systems like WebRTC support simple probing mechanisms [49] that send padding traffic to test for extra bandwidth. For instance, GCC fires a periodic timer (e.g., every 5 s) and injects padding packets to probe capacity. Such coarse probing cannot deliver precise rate control, which requires RTT-scale feedback. As shown in Fig. 21a (Appendix A), GCC behaves almost the same on the alternating link with or without this probing. We show that probing must be synchronized with the congestion controller to be effective.

2.2 Our Solution

Decoupling the encoder from the rate on the wire. As shown in Fig. 2a, a backlogged flow using a state-of-the-art CCA, like Copa, can adapt to network capacity on RTT timescales while also controlling network queueing delay. A key reason is that such CCAs maintain a tight feedback loop, deciding on every RTTand even on every ACKhow many packets to keep in flight [27]. Vidaptive's core idea is to make video stream appear like a backlogged flow to the CCA to leverage their high throughput, low delay, and fast convergence. In Fig. 2, Vidaptive, using Copa for congestion control, achieves the same throughput and network delay as a backlogged Copa. Vidaptive ramps up quickly when bandwidth increases, delivering higher image quality than GCC (Fig. 2b). It also avoids GCCs latency spikes (Fig. 2c) by reacting fast to bandwidth drops.

Vidaptive sends packets on the wire according to the congestion controllers feedback loop. When the encoder overshoots the capacity, Vidaptive queues excess video packets and only sends them when congestion control allows (e.g., based on the congestion window and in-flight packets for window-based CCAs). When the encoder undershoots the available capacity, Vidaptive sends "dummy packets" to maintain the sending rate requested by the controller.[2] This design preserves the CCA's feedback loop, so it continues to behave as if it

[2] This dummy traffic could also be repurposed for helpful information such as forward error correction (FEC) packets [43, 37, 29] or keyframes for faster recovery from loss. We leave such enhancements to

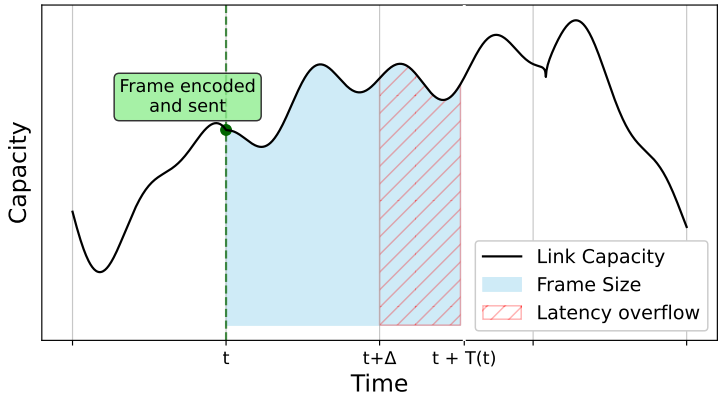


Figure 3 Frame transmission time ($T(t)$) fluctuates with link and encoder variability and can exceed latency thresholds. Vidaptive mitigates this by measuring frame transmission times and adapting the encoders target bitrate online.

were serving a backlogged flow. Our evaluations show that, contrary to the common belief that dummy traffic "seldom works for real-time video" [57], careful use of it in our design improves video bitrate while keeping latency low.

How Vidaptive handles variability. By decoupling the congestion controllers fine-grained feedback loop from the encoders variable output, Vidaptive can accurately track time-varying bottleneck rates independent of video traffic — something GCC and other rate controllers, which rely on video traffic for estimation, cannot do. Still, we must align the video bitrate with the controllers sending rate to ensure good quality and latency.

Despite estimating the link rate accuracy, controlling the frame latency is challenging due to encoder and link rate variability. We show why using a simplified idealized example (Fig. 3). At time t, let the CCA's measured link capacity be $C(t)$. Assume that we set the encoders target bitrate to $C(t)$ and it immediately outputs a frame of size $C(t)\Delta + \sigma(t)$, where Δ is the frame interval and $\sigma(t)$ captures output variability. If no other data is in flight, the frame transmission time $T(t)$ satisfies the following:

$$\int_{t}^{t+T(t)} C(\tau)\, d\tau = \overbrace{C(t)\Delta + \sigma(t)}^{\text{frame size}} \tag{1}$$

Fig. 3 shows frame size as the area under the capacity curve. Network fluctuations can push $T(t)$ beyond acceptable limits. This issue stems from traditional video codecs: once a frame is encoded it cannot be changed. The sender then has two choices: send the oversized frame and cause delay, or drop it and restart with a keyframe, which usually produces an even larger frame [21].

Consequently, it not possible to increase the video bitrate without affecting the tail latency. Pushing the target bitrate too close to link capacity risks latency spikes during sudden capacity drops. For example, a frame that takes 33 ms to transmit on a 2 Mbps link needs 132 ms if the rate suddenly falls to 500 Kbps, leading to an unavoidable increase in latency.

To bound latency, we must reserve headroom and allocate only a fraction of measured

future work and focus solely on the impact of dummy traffic on video congestion control.

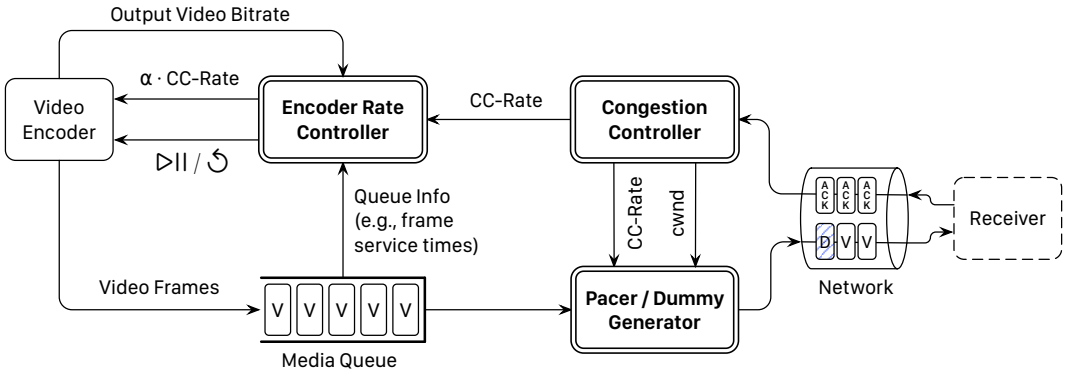


Figure 4 Vidaptive Design. Vidaptive uses a window-based Congestion Controller, Pacer, and a new Dummy Generator to decouple the rate at which traffic is sent on the wire from the encoder. The Encoder Rate Controller monitors frame delays to trigger latency safeguards and picks a new target bitrate as the discrepancy between the `CC-Rate` and the video encoder's current bitrate to control the latency.

capacity to video. Vidaptive does this by setting the encoders target bitrate to $\alpha(t) \cdot C(t)$, where $0 < \alpha(t) \leq 1$. Vidaptive has an online algorithm that tracks frame transmission times and adapts $\alpha(t)$ so the latency stays within a desired bound. Doing so, Vidaptive handles both network and encoder variability (i.e., $\sigma(t)$) simultaneously. Another interesting advantage of this algorithm is that users can tune their desired latency upper bound and allow for a tradeoff between video quality (size of the frames) and latency by tuning α.

3 Vidaptive Design

Fig. 4 shows Vidaptive's overall design. The video encoder encodes frames and sends them to an application-level media queue before sending the packets into the network. At the transport layer, we add a window-based "Congestion Controller," a "Pacer," and a "Dummy Generator" to decouple the rate at which traffic is sent on the wire from what is produced by the encoder, as described in §3.1. We introduce an "Encoder Rate Controller" that monitors the queueing delay frames are experiencing to trigger the latency safeguards described in §3.2. The Encoder Rate Controller also controls the encoder's target bitrate based on the discrepancy between the CCA's sending rate, the video encoder's current bitrate, and the observed frame transmission times (§3.3).

3.1 Transport Layer

Congestion Controller. Vidaptive uses a delay-controlling *window-based* congestion control algorithm to determine when to send packets[3]. The congestion window (*cwnd*) limits the maximum number of in-flight bytes between the sender and the receiver. The sender keeps track of the number of bytes in flight and only sends out new packets if the amount in-flight is less than *cwnd*. Vidaptive can use any CCA that controls delay effectively. Our focus is not on designing a new CCA; fortunately, several excellent algorithms already exist. We use two recently proposed algorithms, Copa [7] and RoCC [4] in our implementation. Most of our results use Copa, which keeps a tight check on the network delay, reacts to

[3] We exclude buffer-filling CCs such as TCP Cubic from our focus due to their tendency to build large queues and increased latency.

changes quickly, and maintains high utilization. Vidaptive's sending behavior is identical to its underlying CCA (§2.2), so it inherits these properties as well as the flow-level fairness characteristics of the CCA.

Pacer. Beside enforcing the *cwnd* constraint, the sender also paces the packet transmissions at the CCA's sending rate (`CC-Rate`), computed as the *cwnd* divided by smoothed RTT (a weighted moving average of RTT samples). Since the encoder's output bitrate may overshoot the available capacity (§2.1), the Pacer is responsible for avoiding a sudden burst of packets.

Dummy Packets. Our goal is to reshape the video traffic as a backlogged flow to enable the CCA to track changes in available bandwidth quickly and accurately (§2.2). To this end, we generate "dummy packets" if the CCA is ready to send a packet as if the stream as backlogged, but no video packets are available in the Pacer. Dummy packets are strictly lower priority and are sent judiciously and with respect to delays; they are only sent if the Pacer queue is empty and congestion control detects free bandwidth. To limit unnecessary traffic, dummy packets are withheld if the next frame is expected soonspecifically, within 25% of the inter-frame time (8.25 ms for 30 FPS) in our implementation. The benefit of this mechanism is that if the network is about to drop, the dummy packets sent a few milliseconds before a frame will cause a higher queuing delay in the network. On the other hand, if the network rate increases, skipping a few milliseconds of traffic does not noticeably slow the controllers reaction.

Finally, dummy packets are disabled once the video reaches its maximum achievable bitrate (12 Mbps in our experiments).[4]. Beyond this point, extra traffic would not improve video quality or reveal useful bandwidth information.

3.2 Safeguarding against Latency Spikes

Encoder Pause. The encoder can overshoot and produce oversized packets (§2.1). To control the latency, Vidaptive monitors how long packets sit in the pacer queue before being sent. If the oldest packet waits longer than a certain pause threshold (τ), we *pause* encoding of frames, and only resume it only once the pacer queue has drained. We set $\tau = 33$ ms by default in our implementation for a camera rate of 30 FPS, thereby pausing encoding if packets from the previous frame remain unsent. The intuition is that there is no point in encoding a new frame that would have to sit in the Pacer queue, waiting for an old frame to finish transmission. Instead, we always encode and transmit fresh frames when they have a high chance of reaching the receiver with acceptable latency. Note that a high delay through the Pacer queue reflects congestion at the bottleneck link. If we ignore `CC-Rate` and transmit the packets stuck in the Pacer queue (as currently implemented in WebRTC), they would still have to wait at the bottleneck link, and clearing them would take longer when the link bandwidth improves.

3.3 Handling Variability

Real-time video must cope with variability from both the network and the encoder. Even if the congestion controller accurately tracks available capacity, frame sizes fluctuate unpredictably, and sudden bandwidth drops can stretch frame latency beyond acceptable limits. To manage this, Vidaptive treats target bitrate selection as a control problem: it chooses a

[4] Our analysis of the 1080p videos within our dataset showed that the peak bitrate typically reached was around 12 Mbps, so for this collection of videos, 12 Mbps represents the upper limit of bitrate.

fraction α of the estimated network rate (`CC-Rate`) to allocate to the encoder (§2.2). A smaller α leaves more headroom, producing smaller frames that drain quickly and keep latency low. A larger α raises frame quality but risks larger latency. In this way, α becomes the knob for navigating the tradeoff between latency and video quality.

Preliminaries. To estimate the frame transmission time, we measure the frame service time: the interval from when the first packet of a frame arrives at the front of the pacer until its last packet is sent out on the network. The frame service time depends on both `CC-Rate` and frame sizes. As a result, it is impacted by fluctuations in both the encoder's output and in the `CC-Rate`. These fluctuations are out of our control and can be viewed as exogenous noise. However, we can influence the *expected* frame sizes by controlling the encoder's *target bitrate*. The crux of our method is to pick the target bitrate in a way that controls the distribution of frame service times. Vidaptive measures the frame service time for each frame in the Pacer.

Consider N frames over a duration D, each with a service time d_i for $i \in 1, 2, \ldots, N$. The distribution of d_i informs us about the frame latencies. If the camera's frame rate is f_{max} (typically 30 FPS), the frame interval is $\Delta = \frac{1}{f_{max}}$ (typically 33 ms). For maximum efficiency, service times should be close to, but less than, Δ, so that the last packet of a frame is sent just as the next frame is encoded. For example, if the 80^{th} percentile of the d_i distribution is larger than Δ, then 20% of the frames take more than a frame interval (see Fig. 3). These frames increase the delay of subsequent frames and may trigger latency safeguard (§3.2), which skips frames. A heavy-tailed distribution of d_i values directly translates into high tail latency of video frames.

Choosing Encoder's Target Bitrate. Suppose the encoder is given a target bitrate $\alpha \cdot$ `CC-Rate`, where $0 < \alpha \leq 1$. A larger α produces larger frames and longer service times d_i, so d_i is function of α. Our goal is to find α to control the latency, or:

$$\text{Percentile}_\lambda \ (d_i \cdot \alpha)_{i=1}^{N} = P \tag{2}$$

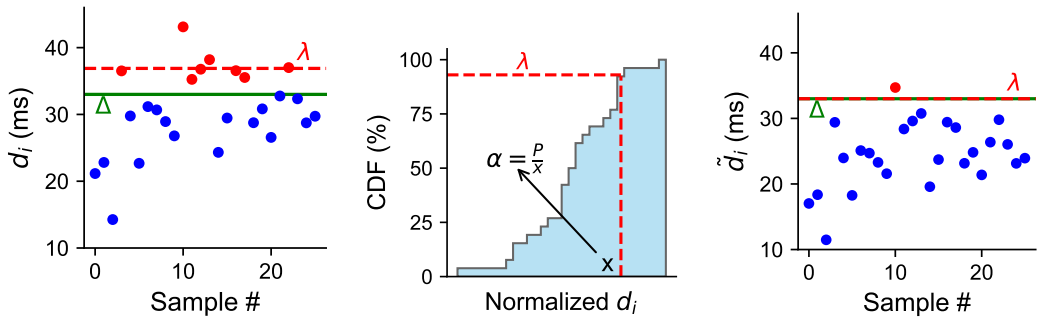
where $\lambda \in \mathbb{R}_{(0,1]}$ is the chosen percentile for latency control. In plain terms, the goal is to align the λ-th percentile of service times with the desired target P. Real-time applications typically use a high λ (e.g., $\lambda = 0.9$) to bound latency tail. For example, $\lambda = 1$ focuses on worst-case latency. The parameter P expresses the tradeoff between quality and latency. Choosing P closer to Δ drives frame service times to Δ, increasing the video bitrate to near capacity. This increases video quality at the cost of potentially higher latency. When P is picked smaller than Δ (e.g., $P = \frac{\Delta}{2}$) the target bitrate values are chosen conservatively, resulting in lower latency and potentially lower quality.

Finding α. Ideally, we would solve Equation 2 over *future* frames. But modeling d_i for unseen frames is difficult because it depends on both future video content (e.g., motion) and future changes in `CC-Rate`. Instead, we choose α based on the observed service times of *recent frames*.

Suppose we have frame service times d_i for $i \in \{1, 2, .., N\}$ measured over the last D seconds. We want to know what value of α would have achieved Equation 2 for these frames? If tr_i is the target bitrate used to encode frame i, had frame i been encoded by $\alpha \cdot$ `CC-Rate` instead, the frame service time would have been

$$\tilde{d}_i(\alpha) = d_i \frac{\alpha \cdot \texttt{CC-Rate}}{tr_i}$$

We refer to $\tilde{d}_i(\alpha)$ as the *counterfactual* service time for frame i. This estimate assumes that frame size is proportional to the target bitrate (and hence proportional to α) and that



(a) Service time samples **(b)** Solving the equation for α **(c)** Counterfactual samples

Figure 5 Finding target bitrate fraction (α). Given a set of frame service time samples (left), whose λ^{th} percentile and outliers higher than $P = \Delta$ are shown in red, we find α that matches the counterfactual λ^{th} percentile to P (middle). We update the counterfactual values of frame service time with α to have fewer outliers above P (right).

changing the target bitrate would not have changed `CC-Rate`.[5] Using these counterfactual service times, we can now find α in Equation 2.

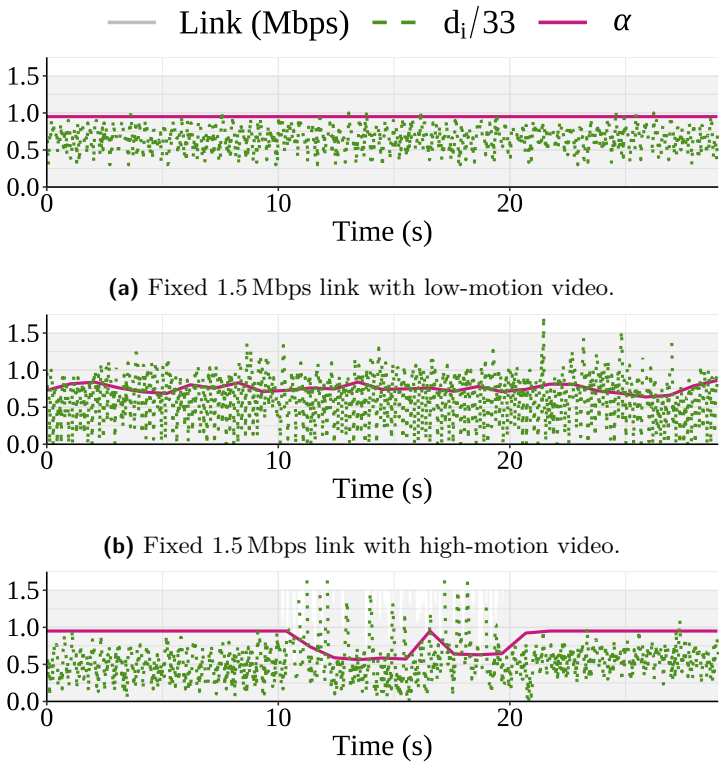
$$\alpha = \min\left(\frac{P}{\text{Percentile}_{\lambda}\left(\{d_i \frac{\texttt{CC-Rate}}{tr_i}\}_{i=1}^{N}\right)}, 1\right) \tag{3}$$

Fig. 5 shows an example of this counterfactual problem for $\lambda = 0.9$. Fig. 5a shows the frame service time samples and their λ^{th} percentile (red line). The samples that are less than $P = \Delta$ are colored in blue, and those larger than P (which would cause higher latency) are colored in red. Fig. 5b shows the distribution of $d_i \frac{\texttt{CC-Rate}}{tr_i}$ (called as normalized d_i) and its λ^{th} percentile (denoted by x). Here, α is set to $\frac{P}{x}$. Fig. 5c shows the counterfactual frame service time values, $\tilde{d}_i$, had α been used to encode them. The scaled-down $\tilde{d}_i$ values reduce the number of outliers above P (decreasing the latency).

How does α respond to variability? α controls the headroom between the encoder's target bitrate ($\alpha \cdot$`CC-Rate`) and the estimated network rate (`CC-Rate`). Vidaptive automatically adjusts α to cope with variability caused by both changing video content and fluctuating network rates. Fig. 6 illustrates this behavior in three scenarios, all with $\lambda = 0.9$ and $P = 33$ ms.

Fig. 6a shows the values of α and the ratio of frame service time to $\Delta = 33ms$. In steady conditions, α has a stable value of ~ 1 and the P90 of the frame service times is below 33 ms. Next, we test Vidaptive on the same link with a high-motion video [54] in Fig. 6b. The encoded frame sizes and the frame service times are now much more variable. Consequently, α adapts to the variable encoder output by switching to a lower value to control the tail of frame service times. Finally, we run Vidaptive on a 1.5 Mbps link that experiences 10s of high variability in Fig. 6c. Before the fluctuations start at 10s, frame service time values are well below Δ with $\alpha \sim 1$. During the noisy period (10s–20s), when the frame service times increase, α decreases to improve the P90 of frame service time and reduce video bitrate. When the link steadies after the 20s, α resets to its high value.

[5] This holds because Vidaptive decouples `CC-Rate` from the encoder.



(a) Fixed 1.5 Mbps link with low-motion video.

(b) Fixed 1.5 Mbps link with high-motion video.

(c) 1.5 Mbps link with 10s fluctuation and a low-motion video.

Figure 6 α's response to link and video encoder variations. α picks lower values (lower target bitrate) when the link capacity or encoder output varies significantly to maintain good control over the frame service times. d_i denotes the frame service time in milliseconds.

4 Implementation

We implemented our system on top of Google's WebRTC [48].

Congestion Controller. We implement two window-based, delay-sensitive algorithms: Copa [7] and RoCC [4], a BBR variant. For Copa, we reuse the logic from its reference implementation [6]. RoCC works by fixing the congestion window (*cwnd*) to slightly more than the number of bytes received in the last $(1+\gamma)\text{RTT}min$ interval, where RTTmin is the minimum observed round-trip time. This ensures a queuing delay of $\gamma \cdot \text{RTT}_{min}$, with γ controlling delay sensitivity. Unless otherwise noted, we use Copa as the default controller.

Dummy Generator. We repurpose the padding generator in WebRTC to generate dummy packets that are within the *cwnd* and smaller than 200 bytes each. Dummy packets are ACKed by the receiver (via RTP/RTCP feedback), but their payload is discarded. The dummy packets have "padding" type in their packet header, and the receiver distinguishes them from the "video" packets.

Latency Safeguards. If the oldest packet in the pacer queue waits longer than the pause threshold (τ in §3.2), the transport layer sets the encoder target bitrate to zero to signal a *pause*. We reuse WebRTC's support for buffering the latest unencoded camera frame until transmission resumes.

Encoder Rate Controller. Vidaptive records the target bitrate used for encoding each frame. It also measures frame service time by recording the time that the first packet of a frame is at the front of the pacer and when the last packet of the frame leaves the pacer

(this does not include the queuing time the frame spent in the pacer before frames ahead of it). Every second ($D = 1$ s), Vidaptive updates α based on these service times using the algorithm in §3.3. We also apply an EWMA smoothing over the values of α to ensure gradual changes in the encoder's target bitrate over time.

5 Evaluation

We evaluated Vidaptive on Mahimahi [39] and Pantheon [52] emulated links. Our experimental setup is detailed in §5.1, and we use this to benchmark against existing systems in §5.2. A thorough examination of Vidaptive's design features is presented in §5.3. Additional evaluations focusing on less critical parameters (λ and τ) and results with alternative quality metrics are available in the appendix.

5.1 Setup

Testbed. We built a testbed, implemented in C++, on top of WebRTC [16] that runs a headless peer-to-peer video call between two endpoints. The sender program emulates a camera by reading video frames from an input file and the receiver records the received frames to an output file. To match video frames between the sender and the receiver for visual quality and latency measurements, a unique 2D barcode is placed on each frame [21]. We emulate different network conditions between the sender and receiver by placing the receiver behind a Mahimahi shell. All experiments are run for 2 min on a link with a one-way delay of 25 ms.

Metrics. To assess the performance of Vidaptive, we focus on two key metrics: frame quality and frame latency. For frame quality, we use Video Multimethod Assessment Fusion (VMAF [38]), comparing received frames to their original sources. Additionally, we report Structural Similarity Index Measure (SSIM [47]), and report the Peak Signal-to-Noise Ratio (PSNR [24]) in Appendix Fig. 24. We measure frame latency by the duration from *frame read* at the sender to *frame display* at the receiver. We set the display time for undelivered frames to the presentation time of the next frame displayed [21]. In addition, we report the total video bitrate and frame rate at the receiver.

Network Traces. We evaluate each scheme on a set of 15 cellular traces bundled with Mahimahi [39] and also use synthetic traces to illustrate the convergence behavior in Appendix C. For the cellular traces, we use a large bottleneck buffer size that does not incur packet loss. We also evaluate each scheme on three Pantheons calibrated network emulation traces [52]. These traces were tuned to mimic real Internet paths. They have limited buffers (and thus incur congestion-induced loss) as well as stochastic packet loss.

Videos. We use a dataset of 20 different 1080p videos, each playing at 30 frames per second. The dataset is curated from YouTube and spans different levels of motion intensity across sports, cooking, video conferencing, nature documentaries, live performances, and screen sharing. Unless specified otherwise, all experiments are conducted on the complete set of videos from this dataset. Audio is disabled throughout the experiments. See Tab. 1 in appendix for detail.

Baselines. We benchmark Vidaptive against several established congestion control algorithms to highlight its performance for real-time video over variable links. First, we consider the Google Congestion Control (GCC) algorithm, the standard for many real-time applications [11, 48]. The GCC protocol has changed since publication [11]; we used the latest implementation in the M108 [16] version of WebRTC. Additionally, we compare Vidaptive to Performance-oriented Congestion Control (PCC), another rate-based approach [17]. We

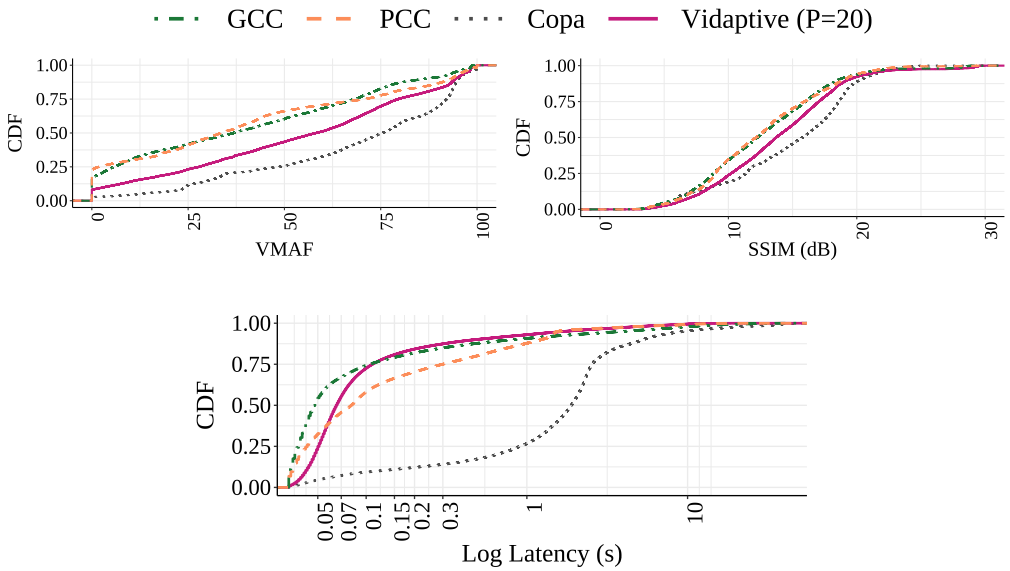


Figure 7 CDF of frame VMAF and SSIM and latency across all the 1M frames. Compared to GCC and PCC, Vidaptive achieves higher VMAF and SSIM on all percentiles while getting better latency on higher percentiles. Vidaptive increases the median latency over GCC by a marginal amount of 17 ms. Copa's higher VMAF scores come at a great latency cost, rendering it unsuitable for real-time video applications.

include analysis against Copa, a state-of-the-art delay-based congestion control algorithm [7]. All algorithms are implemented within the same framework.

5.2 Vidaptive improves Video Quality and Tail Latency.

Cellular Traces. In Fig. 7, we compare Vidaptive against GCC, PCC, and Copa on the cellular traces bundled with Mahimahi [41]. The figure shows cumulative distribution functions (CDFs) for VMAF, SSIM, and frame latency across all traces and frames. Compared to GCC and PCC, Vidaptive consistently delivers higher visual quality. It raises the average VMAF to 53.5% (vs. ∼39% for GCC and PCC) and the median (P50) VMAF to 58.2% (vs. 36.8% for GCC and 33.4% for PCC). For SSIM, Vidaptive improves the average to 14 dB (vs. 12 dB) and the median to 13.8 dB (vs. 12 dB). These quality increase is thanks to Vidaptive's effective link utilization.

Although Copa achieves higher VMAF and SSIM scores than Vidaptive, it suffers from significantly higher latency (e.g., an average of 2.9 seconds), making it impractical for real-time video. The issue is that, with video traffic, Copa is application-limited for most of the time. Copa is designed for backlogged flows and does not interact well with video streams: during periods of video silence, it observes no queuing delay and grows its congestion window to its maximum cap, even though no packets are in flight. When the network rate later drops, this inflated window causes excessive queuing and delay. In contrast, Vidaptive also uses Copa as its underlying CCA, but reshapes the video source to behave backlogged, thus preserving Copa's responsiveness and effectively controlling latency.

Vidaptive significantly improves tail frame latency, particularly beyond the 75^{th} percentile. These higher percentiles correspond to conditions with severe link variability and outages, where Vidaptive reacts faster thanks to its tight congestion-control feedback loop. At the 95^{th} percentile, for example, Vidaptive reduces frame latency by 2246 ms relative to GCC, cutting it from 3941 ms to 1695 ms. It also improves average latency by 351 ms

(383 ms vs. GCCs 734 ms). Compared to PCC, Vidaptive lowers both the average latency (383 ms vs. 521 ms) and the median latency (65 ms vs. 81 ms). Vidaptive has a slightly higher median latency than GCC (65 ms vs. 48 ms), caused by Vidaptive s always-on probing, which maintains a small standing queue to preserve tight feedback. Prior work shows that latencies well under 100 ms are generally imperceptible to users [14, 15], so this small increase has little practical impact.

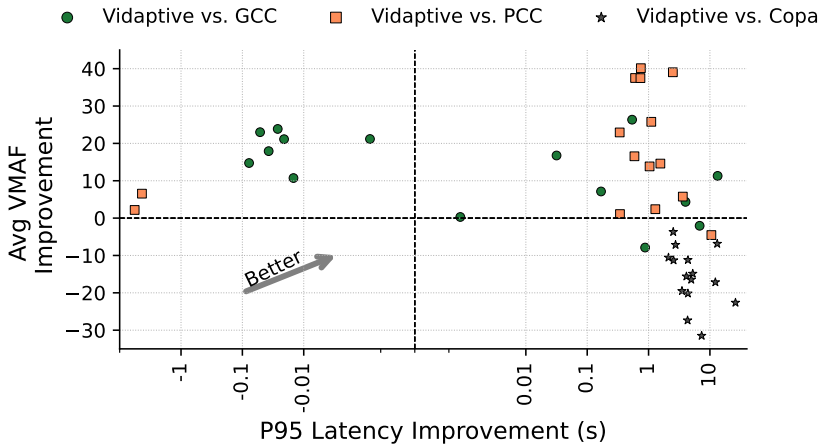


Figure 8 Performance benefits of Vidaptive over GCC, PCC, and Copa. Vidaptive either improves latency or quality or both over all baselines.

Fig. 8 summarizes the tradeoffs between video quality and latency by plotting the average VMAF and P95 frame latency of Vidaptive across all videos for each cellular trace. Each point corresponds to a trace. Overall, Vidaptive consistently improves either quality, latency, or both compared to all baselines.

Against GCC, Vidaptive delivers higher VMAF in 80% of traces and reduces P95 latency in half of them; in the remaining half, it increases P95 latency by less than 100 ms, a negligible penalty given the quality gains [15]. Compared to Copa, Vidaptive reduces P95 latency by about one second across all traces. Relative to PCC, Vidaptive lowers P95 latency in 86% of traces (13 out of 15) and achieves higher VMAF in 14 out of 15 traces.

Pantheons Calibrated Emulators. We further evaluate Vidaptive on three calibrated emulated paths from Pantheon [52], which approximate real Internet routes. Unlike the Mahimahi cellular traces, these paths have constant capacity with Poisson packet delivery, but introduce loss and shallow tail-drop queues. We test on three representative traces“Nepal to AWS India (Wi-Fi)” [3], “Mexico to AWS California (Cellular)” [2], and “AWS Brazil to Colombia (Cellular)” [1]using five videos.

Fig. 9 shows that Vidaptive achieves substantially higher quality than GCC, with a median VMAF of 44% vs. 15% and an average of 40% vs. 29%, while both reach a P95 of 79%. Against PCC, Vidaptive achieves higher VMAF across all percentiles.

In terms of latency (Fig. 9), Vidaptive reduces P95 latency by 933 ms (1739 → 806) and average latency by 89 ms (362 → 273) compared to GCC, at the cost of a modest 19 ms increase in median latency (137 → 156). This small rise is consistent with our cellular-trace results and reflects the shallow standing queue that Copa, Vidaptive s underlying CCA, maintains to track bandwidth. Compared to both PCC and Copa, Vidaptive achieves strictly lower latency at all percentiles.

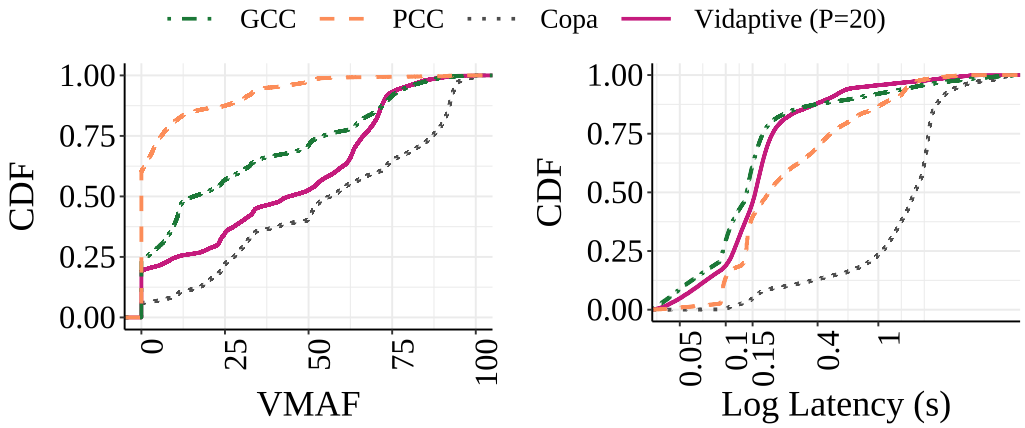


Figure 9 CDF of VMAF and latency for all the frames in Pantheon traces. Vidaptive improves the quality over PCC. Compared to GCC, Vidaptive has better quality in low-medium percentiles and similar quality in high percentiles. Copa has higher quality at the cost of significantly higher latency. Vidaptive improves the latency in all percentiles vs. Copa and PCC. Compared to GCC, Vidaptive improves 95^{th} by $\sim 1\,\text{s}$ while increasing the median latency 19 ms vs. GCC.

5.3 Vidaptive navigates Quality and Latency Tradeoff.

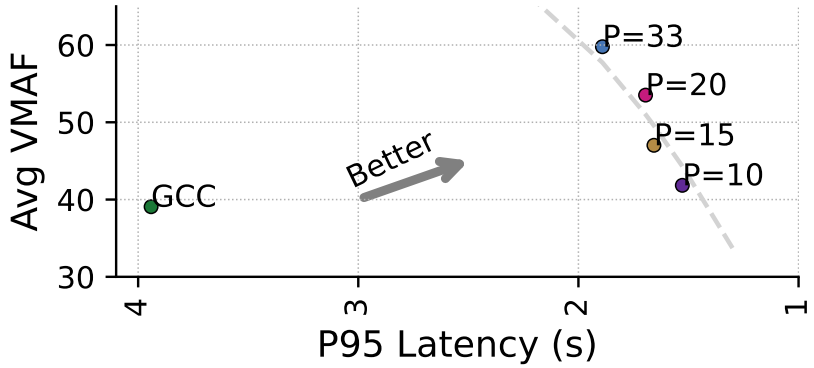
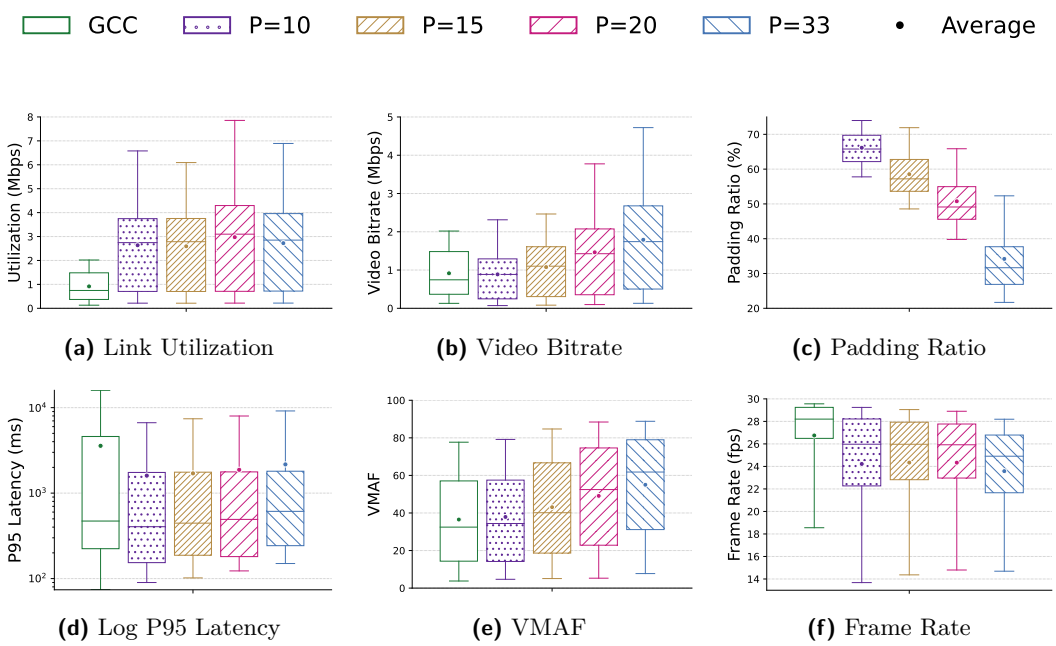


Figure 10 Average VMAF vs P95 Latency over all the frames. Vidaptive trades off latency and quality by parameter P and offers a better frontier than GCC.

In our analysis of Vidaptive, we first focus on the impact of varying the desired frame service time (P), which determines the tradeoff preference of latency and quality (§3.3). Fig. 10 presents the average VMAF vs. P95 of latency of GCC and Vidaptive using different values of P over all the frames. This illustrates that Vidaptive achieves a better performance frontier than GCC. Vidaptive also offers different operating points for latency and quality tradeoff.

To provide additional insight, Fig. 11 shows the distribution of metrics across all traces and videos (15×20 settings). For the remainder of the paper, we focus on Mahimahis cellular traces.

A key property of Vidaptive is that dummy traffic decouples link utilization from the video encoders decisions. As a result, varying P does not impact overall utilization, which remains consistently higher than GCC (Fig. 11a). By increasing P, Vidaptive chooses higher video bitrates (Fig. 11b) and quality (Fig. 11e), which correspondingly reduces the fraction of dummy traffic (Fig. 11c). However, higher P values also push Vidaptive to prioritize



(a) Link Utilization **(b)** Video Bitrate **(c)** Padding Ratio

(d) Log P95 Latency **(e)** VMAF **(f)** Frame Rate

Figure 11 Performance comparison of GCC and Vidaptive with different values of target frame service time P. Changing P does not affect link utilization since Vidaptive decouples utilization from video. Increasing P raises video bitrate and reduces padding but also increases P95 latency and frame skipping. Whiskers indicate the 5^{th} and 95^{th} percentiles (P5, P95); boxes show the interquartile range (P25–P75); the horizontal line marks the median (P50); and the bullet (•) denotes the mean.

video bitrate over latency, raising P95 latency (Fig. 11d)though it still remains lower than GCC. The frame rate remains mostly stable, but at $P = 33\,\text{ms}$, frame skipping becomes more frequent, slightly reducing frame rate (Fig. 11f).

Looking at the tension between the video and padding bitrates, an important question arises: Can we reduce dummy traffic by increasing the video bitrate, without paying a latency penalty? The answer depends on the variability of the underlying link.

To illustrate, consider an idealized setting. Suppose the transport layer has perfect, instantaneous knowledge of link capacity $C(t)$, and the encoder is ideal, producing noise-free frame sizes immediately (i.e., $\sigma(t) = 0$ in Equation 1). If the encoder target bitrate is $\alpha C(t)$, for a fixed α, it generates a frame of size $\alpha C(t)\Delta$, where $\Delta = 33ms$ is the frame interval. The frames transmission time, $T(\alpha, t)$, is:

$$\int_t^{t+T(\alpha,t)} C(\tau)\, d\tau = \overbrace{\alpha\, C(t) \cdot 33\,\text{ms}}^{\text{frame size}} \tag{4}$$

Given a link trace, we know $C(t)$. Using this, we computed $T(\alpha, t)$ for $\alpha \in \{0.5, 0.75, 0.9, 1.0\}$ over $t \in [0, 2\,\text{min}]$ across all traces. Fig. 12 plots the distribution of P95 values of $T(\alpha, t)$, where each bar aggregates all t and all traces for a fixed α. As α increasesi.e., as the encoder consumes a larger fraction of available capacitythe P95 of T also rises. This reveals a fundamental property: even with perfect knowledge of $C(t)$ and an ideal encoder, tail latency is lower-bounded by the links intrinsic variability.

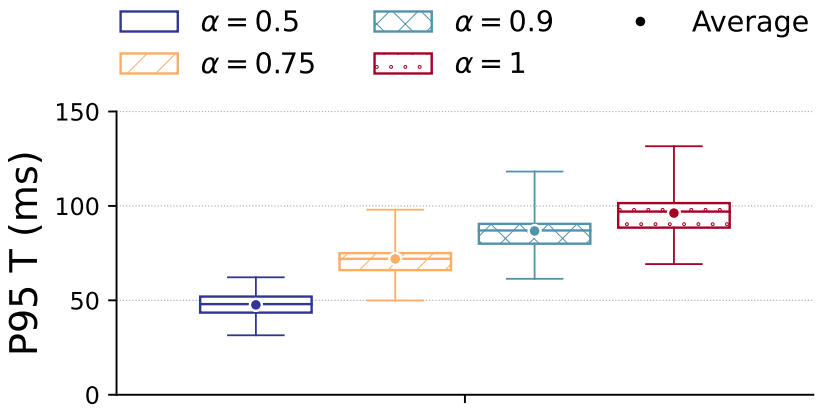


Figure 12 Relationship of target bitrate factor α and the 95$^{\text{th}}$ percentile (P95) of frame transmission time T, with an ideal encoder. Larger α values allocate more link capacity to video but shift P95 latencies higher. This illustrates an intrinsic tradeoff between maximizing video bitrate and controlling latency under highly variable network conditions.

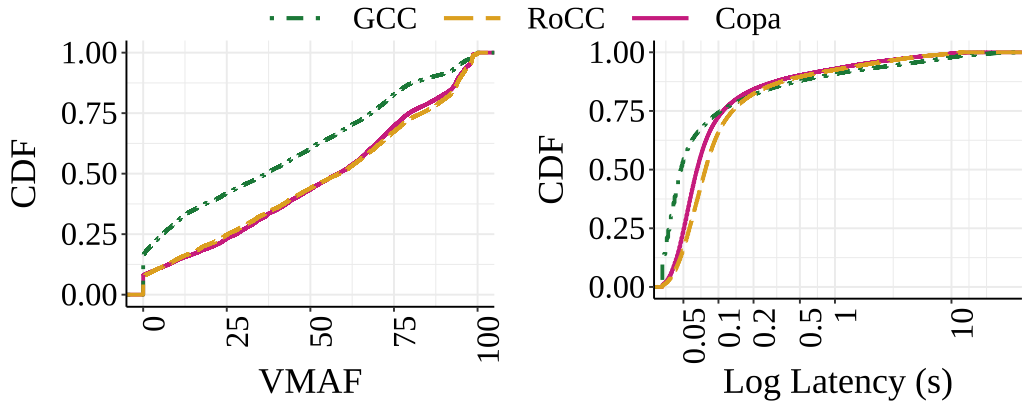
Consequently, to bound tail latency, Vidaptive must adopt smaller α values on highly variable links, which in turn reduces the video bitrate. Because overall link utilization remains the same under the tight feedback loop (Fig. 11a), Vidaptive compensates by filling the remaining headroom with dummy traffic (Fig. 11c). This dummy traffic allows the CCA to track capacity accurately, but since it is sent only when no video frames are ready, it has negligible effect on frame latency (§3.1). For example, Fig. 12 shows to keep the frame transmission times below 50 ms, α must be set to 0.5, which translates into 50% utilizing the link with video and 50% with padding traffic. Under such highly variable link conditions, and given strict latency thresholds, dummy traffic cannot be reduced beyond a certain limitdoing so would compromise accurate link estimation. We find this quite interesting, that even with the cost of high padding ratios, Vidaptive is able to achieve better video quality and latency compared to GCC thanks to the tight feedback loop of the congestion controller.

5.4 Understanding Vidaptive's Design

Using a Different Congestion Controller. To show that Vidaptive can work with other delay-sensitive window-based CCA, we replaced Copa with RoCC [4]. Fig. 13a shows the VMAF and Fig. 13b shows the latency CDFs of Vidaptive (RoCC) compared to GCC. Vidaptive (RoCC) follows similar trends as Vidaptive (Copa) and improves the quality in all percentiles compared to GCC. Compared to GCC, Vidaptive (RoCC) increases the median latency from 48 ms to 77 ms while reducing the P95 of latency from 3.9 s to 1.9 s. Compared to GCC, Vidaptive (RoCC) increases the median VMAF from 36 to 56 and the average VMAF from 39% to 53%. Copa overall has a lower latency than RoCC, so we choose it as the default CCA in Vidaptive.

Ablation Study. To understand the impact of different components in Vidaptive's design, we evaluate the benefits of adding dummy traffic at the transport layer (§3.1), having the latency safeguards (§3.2), and running the encoder bitrate selection approach described in §3.3. Fig. 18 shows the distribution of metrics over all the traces and five videos for different system variations. P is selected as 33 ms for the rest of experiments.

In "Copa," we replace GCC with a window-based congestion control algorithm but keep the remaining modules similar to GCC. Copa is more aggressive than GCC in utilizing



(a) VMAF CDF **(b)** Latency CDF

Figure 13 CDF of frame VMAF and latency across all the 1M frames. Compared to GCC, Vidaptive (RoCC) achieves higher VMAF while getting lower tail latency. Vidaptive (RoCC) has a similar performance to Vidaptive (Copa).

bandwidth, improving the average link utilization and video bitrate by almost 3×. However, this aggressiveness causes an average P95 latency of *9 seconds*. The frame rate for all schemes using Copa as CCA reduces because Copa's window-based mechanism, unlike GCC, blocks frame transmissions when network is congested.

In "Vidaptive w/o Dummy," we removed dummy traffic (§3.1) from Vidaptive. Fig. 15 shows that removing the dummy traffic reduces the average *video* bitrate from 2 Mbps to 1.4 Mbps and its P95 from 5.1 Mbps to 2.8 Mbps. However, the average P95 latency and the frame rate do not change significantly compared to Vidaptive: the average P95 latency decreases by only 5.6% when dummy traffic is removed (from 1894 ms to 1788 ms). This slight increase results from CCA keeping a small queue in the network when it uses dummy traffic that causes a bigger spike (and P95 latency) when the network degrades abruptly. This result indicates that the dummy traffic has a negligible impact on latency but it improves the video bitrate by helping CCA maintain its loop to discover bandwidth faster.

In "Vidaptive w/o Latency," we disable the latency safeguards on top of Vidaptive. Disabling latency knobs hurts the P95 latency (Fig. 16) and increases it compared to Vidaptive (1894 ms vs. 6087 ms on average and 8.6 s vs. 26 s in P95). Since the latency safeguards pause encoding of frames, the overall frame rate and video bitrate increases without them.

In "Vidaptive w/o Bitrate Selection," we disable the encoder bitrate selection of Vidaptive (module choosing α in §3.3). Without this mechanism, the CCA's link estimate is directly given to the encoder, which results in a higher video bitrate and latency in highly variable links. When `CC-Rate` is fed directly to the encoder, it attempts to produce frames that take 33 ms to be sent at the current link rate. However, if the link rate drops, such an encoded frame takes more than 33 ms to be transmitted. This, in turn, triggers the latency safeguards that drop the next frames to prevent the latency from increasing. As a result, the frame rate in "Vidaptive w/o Bitrate Selection" is lower than Vidaptive.

Finally, in "Vidaptive", the system aims to find the right target video bitrate for the encoder as described in §3.3. Because this system balances frame latency and frame quality, the video bitrate is slightly lower than "Vidaptive w/o Latency" and "Copa" but achieves better latency. The utilization is comparable across all schemes with dummy traffic since it pads any encoder output to match the link rate.

Another system variation is disabling the dummy traffic and removing the pacer. In-

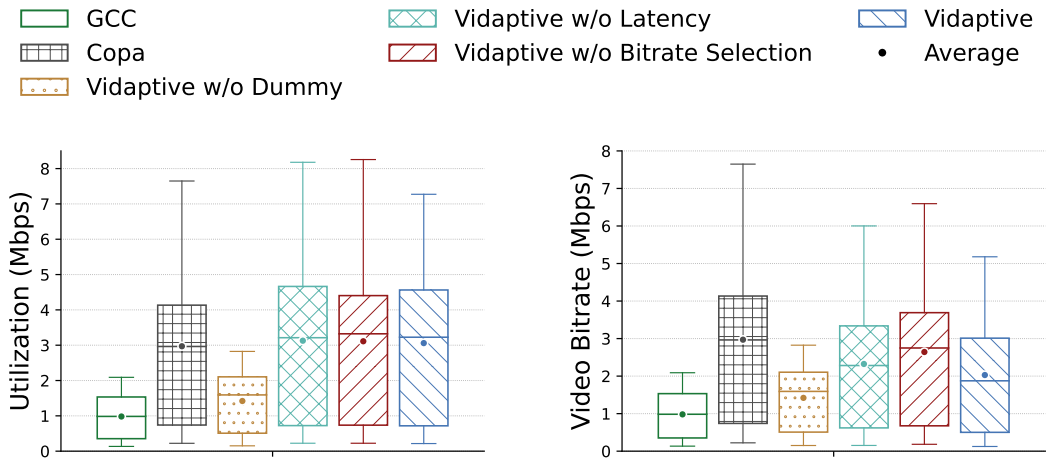
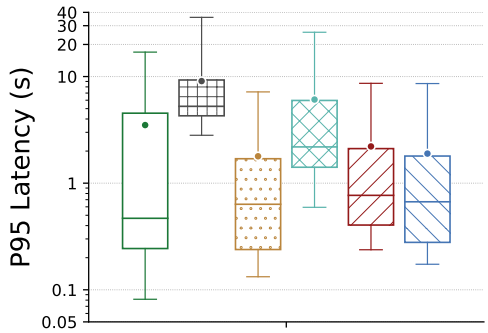


Figure 14 Link Utilization

Figure 15 Video Bitrate



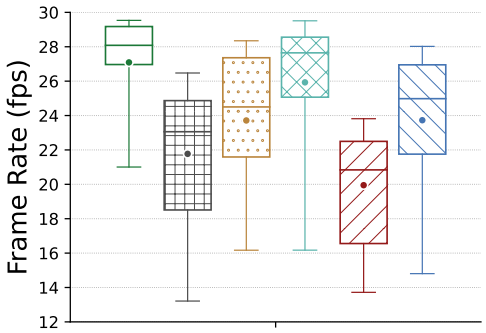


Figure 16 Log of P95 Latency

Figure 17 Frame Rate

Figure 18 Performance comparison of GCC and different Vidaptive components. "Copa" improves video bitrate and utilization but hurts frame latency. Dummy traffic improves video bitrate. "Vidaptive w/o Latency" shows that latency knobs reduce the latency by *seconds*. Vidaptive has higher frame rate and lower latency by using encoder bitrate selection than without it. Since schemes with Copa as CCA do not send frames in outages, they have lower frame rate than GCC.

stead of controlling the frame service time, this system applies the bitrate selection scheme described in §3.3 on the measured one-way frame latencies. We call this "Latency Optimizer". It picks the next target bitrate based on the measured frame latencies and corresponding target bitrates. Fig. 19 compares the Latency Optimizer, Vidaptive, and GCC for $\lambda = 0.5$ and $\lambda = 0.9$. For $\lambda = 0.9$. "Latency Optimizer" opts to control P90 of measured latencies, and although the latency values are low, the quality drops significantly even compared to GCC. For $\lambda = 0.5$, "Latency Optimizer" targets a median latency of 33 ms, but the tail latency increases compared to Vidaptive and GCC without meaningful gains in quality. This approach fails because bitrate selection alone reacts too slowly to rapid link variations (issue similar to §2.1): it relies on latency samples that arrive only once every 33 ms, making it sluggish to raise the bitrate when bandwidth increases and equally slow to cut bitrate during sharp capacity drops.

Comparison with Salsify. Salsify's design takes advantage of a specialized video codec that allows it to discard already-encoded frames when necessary. Since the peak performance

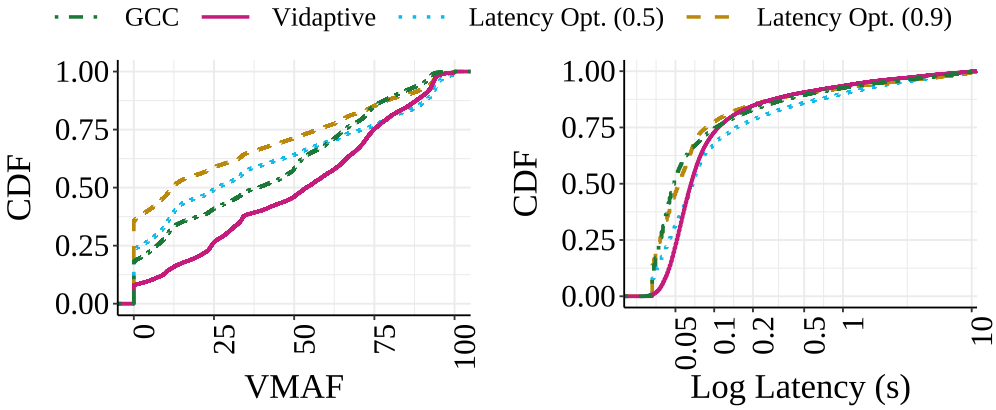


Figure 19 CDF of VMAF and latency for all the frames. The alternative latency optimizer scheme performs worse than Vidaptive.

of this encoder was 24 FPS at 720p, we focus specifically on videos of this size and frame rate. Fig. 20 shows the CDF of VMAF and latency across all the frames. Although Salsify exhibits a higher quality (Fig. 20a and Fig. 20c), it also experiences much higher latency compared to Vidaptive (Fig. 20b). The median latency for Salsify is 119 ms, and the average latency is 1211 ms across all the frames [6]. However, Vidaptive has a median latency of 67 ms, 77 ms, 117 ms and average latency of 468 ms, 535 ms, and 622 ms for $P = 20, 33, 66$ respectively. Fig. 20d shows the P95 latency distribution across all the videos and traces. Salsify experiences *seconds* higher P95 latency than Vidaptive.

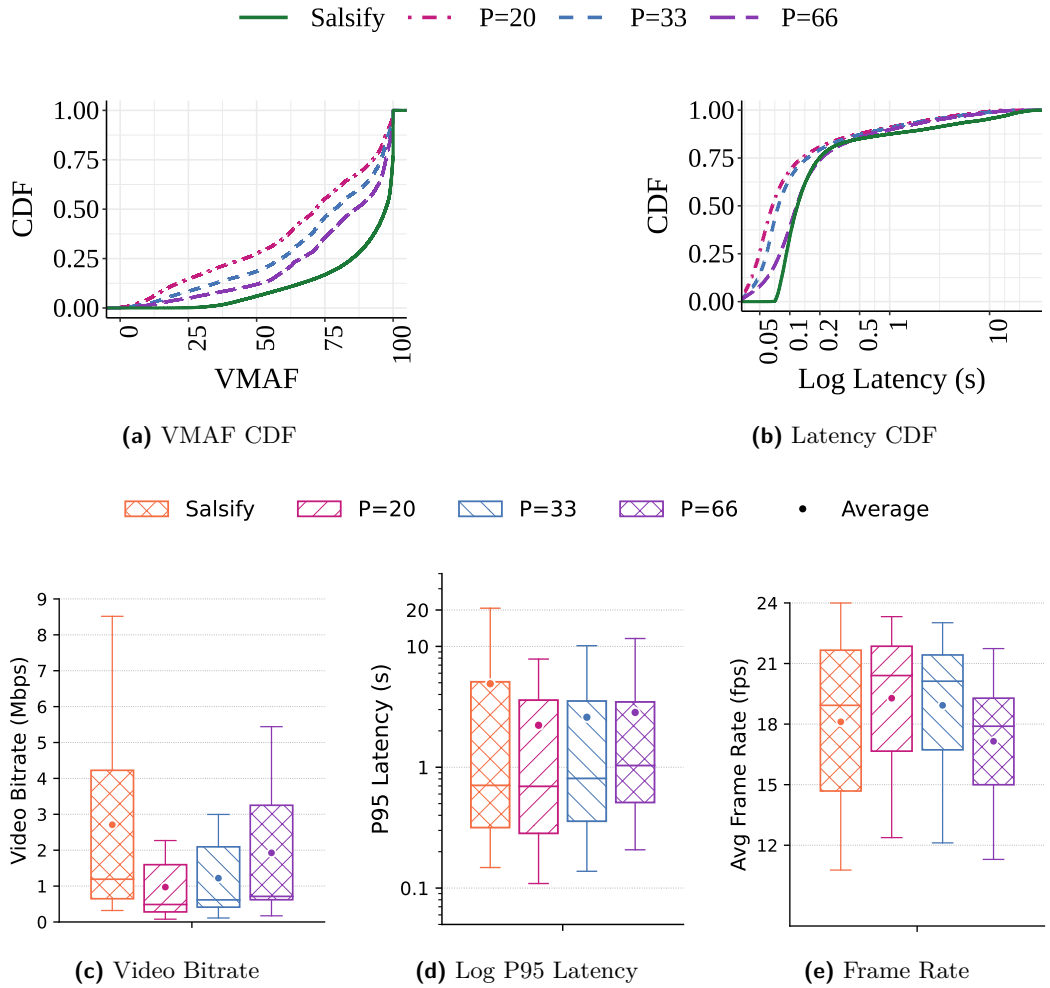
Fig. 20e shows the frame rate for Salsify and Vidaptive. Salsify's primary means of latency control is dropping frames, which manifests itself in lower frame rate. As P increases in Vidaptive, we also have to skip more frames to keep the latency in control.

We also evaluated the effect of different parameter choices in Appendix B. As λ (the latency percentile used in §3.3) increases, both video bitrate and latency decrease. Increasing the pacer pause threshold (τ in §3.2) leaves the video bitrate unchanged, but raises the P95 latency and lowers the frame rate.

6 Related Work

Congestion Control. End-to-end congestion control approaches can be broadly categorized into *delay-based* [8, 7, 31, 36, 28, 55, 11] or *loss-based (buffer-filling)* schemes [17, 23, 20]. Delay-based protocols minimize queuing by adjusting their rate based on measured queuing delay [44, 28, 8] or delay gradients [11, 55, 36]. Loss-based protocols send at the highest rate possible until packet loss or congestion signals are observed. Some schemes, such as BBR [10], use model-based bandwidth and delay estimation and do not fit neatly into either category. Hybrid approaches like Nimbus [22] switch between delay-based and loss-based modes to improve fairness while maintaining high utilization. Congestion control for application-limited flows [19, 26] is particularly relevant to interactive video traffic, which is generated at intervals tied to the frame rate. Schemes such as GCC [12], Pudica [46], Mowgli [5], SCReAM [30], NADA [60], SQP [42], and RL-based approaches [58, 56, 57] are specifically designed for real-time video.

[6] We are evaluating Salsify on a larger set of traces than Salsify's paper.



(a) VMAF CDF **(b)** Latency CDF

(c) Video Bitrate **(d)** Log P95 Latency **(e)** Frame Rate

Figure 20 Comparison of Salsify and Vidaptive across all traces and videos. Salsify exhibits higher average P95 latencies and lower frame rate. Although Salsify can achieve higher quality, it incurs substantially higher latency than Vidaptive.

Video Systems. Many real-time video applications rely on Web Real-time Communication (WebRTC) [48], whose built-in controller GCC [12] adjusts rates using delay gradients. While effective at avoiding queues, GCCs conservative ramp-up and sensitivity to encoder variability often cause under-utilization or latency spikes. Salsify [21] tackled this mismatch by modifying codecs to generate multiple frame versions and picking one that fits current capacity, but codec changes limit deployability. Vidaptive achieves similar alignment of encoder output and network capacity without codec modifications. Adaptive bitrate algorithms [34, 25, 53, 45, 51, 57, 59] solve a related problem for on-demand video, but they operate under more relaxed latency constraints and are not directly suitable for real-time interactive applications

7 Discussion

Tight feedback loop drives responsiveness. We showed responsiveness does not come from specialized controllers, but from preserving RTT-scale feedback. Maintaining this loopeven with paddingenables fast and more accurate bandwidth estimation.

Vidaptive generalizes beyond WebRTC. Although our prototype is built atop WebRTC, the idea of reshaping an application-limited flow into a backlogged one is stack-agnostic, and can be extended to cloud gaming, remote desktop, and XR where variability disrupts feedback loops.

Dummy traffic is both a strength and can be a limitation. Padding preserves the tight control loop behind Vidaptives latency gains, but it consumes bandwidth and may drain device battery. Operators must weigh these costs against qualitylatency benefits. Future work could provide knobs to bound dummy traffic or schedule it judiciously based on the rate controller's uncertainty about the network state. Padding could also be repurposed for FEC, recovery keyframes, or telemetry, reducing overhead.

The latencyquality frontier. Even with perfect link knowledge, encoder and network variability impose limits on tail latency. Vidaptive exposes a new frontier and a tunable tradeoff between quality and latency, but breaking through this frontier may require tighter codeccongestion control integration.

8 Conclusion

This paper introduces Vidaptive, a new real-time video rate control framework that responds quickly to network variability without requiring encoder or hardware modifications. Our evaluation shows that Vidaptive improves video quality while lowering tail latency, outperforming existing controllers like GCC. Vidaptive uses a delay-based CCA as the link rate estimator. To let the CCA operate effectively, it reshapes application-limited video traffic into a backlogged flow by injecting "dummy" packets. It also monitors frame latencies to adjust the encoders target bitrate online to match the video bitrate to the link capacity. A key lesson from this work is that maintaining a tight RTT-scale feedback looppreserved through paddingis essential for responsiveness and latency control.

References

1. Pantheon of Congestion Control: Calibrated emulator for the real path from AWS Brazil to Colombia cellular. https://pantheon.stanford.edu/result/6849/, 2020. (accessed on 2024-02-01).
2. Pantheon of Congestion Control: Calibrated emulator for the real path from Mexico cellular to AWS California. https://pantheon.stanford.edu/result/6848/, 2020. (accessed on 2024-02-01).
3. Pantheon of Congestion Control: Calibrated emulator for the real path from Nepal to AWS India. https://pantheon.stanford.edu/result/6847/, 2020. (accessed on 2024-02-01).
4. Anup Agarwal, Venkat Arun, Devdeep Ray, Ruben Martins, and Srinivasan Seshan. Automating network heuristic design and analysis. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 8–16, 2022.
5. Neil Agarwal, Rui Pan, Francis Y Yan, and Ravi Netravali. Mowgli: Passively learned rate control for {Real-Time} video. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 579–594, 2025.
6. Venkat Arun. Copa's Implementation. https://github.com/venkatarun95/genericCC/blob/master/markoviancc.cc.
7. Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, 2018.
8. Lawrence S. Brakmo and Larry L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. *IEEE Journal on selected Areas in communications*, 13(8):1465–1480, 1995.
9. Dean Bubley. 2015 Q1 Update: WebRTC Market Status & Forecasts Report. https://disruptivewireless.blogspot.com/p/blog-page_30.html, 2015. Accessed: 2024-02-02.
10. Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):20–53, 2016.
11. Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.
12. Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.
13. Cisco. Cisco visual networking index: Forecast and trends, 20172022. https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf, 2019. Accessed: 2025-09-25.
14. Andy Cockburn, Carl Gutwin, and Saul Greenberg. How much faster is fast enough? user perception of latency & latency improvements in direct and indirect touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 2291–2300. ACM, 2015. doi:10.1145/2702123.2702300.
15. Andy Cockburn, Carl Gutwin, Philip Quinn, and et al. Are 100 ms fast enough? characterizing latency perception thresholds in mouse-based interaction. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI)*, pages 433–444. ACM, 2017. doi:10.1145/3025453.3025591.
16. Webrtc developers. https://webrtc.googlesource.com/src/+/a2f5d45b81c6ae5632af0c4c45e8988f330af7f1, December 2022.
17. Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.
18. Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. Pcc vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2018)*, pages 343–356, 2018. URL: https://www.usenix.org/system/files/conference/nsdi18/nsdi18-dong.pdf.

19 G. Fairhurst, A. Sathiaseelan, and R. Secchi. Updating tcp to support rate-limited traffic. RFC 7661, Internet Engineering Task Force, October 2015. URL: https://www.rfc-editor.org/info/rfc7661.

20 Sally Floyd, Tom Henderson, and Andrei Gurtov. Rfc3782: The newreno modification to tcp's fast recovery algorithm, 2004.

21 Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 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)*, pages 267–282, 2018.

22 Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. Elasticity detection: A building block for internet congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 158–176, 2022.

23 Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):6474, jul 2008. doi:10.1145/1400097.1400105.

24 Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

25 Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

26 Jana Iyengar. Cubic Quiescence: Not So Inactive. https://www.ietf.org/proceedings/94/slides/slides-94-tcpm-8.pdf, 2015.

27 Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

28 Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490–2501. IEEE, 2004.

29 Mathias Johanson. Adaptive forward error correction for real-time internet video. In *Proceedings of the 13th Packet Video Workshop, Nantes, France*, pages 1–9. Citeseer, 2003.

30 Ingemar Johansson and Zaheduzzaman Sarker. Self-Clocked Rate Adaptation for Multimedia (SCReAM). RFC 8298 (Experimental), December 2017.

31 Changhyun Lee, Chunjong Park, Keon Jang, Sue B Moon, and Dongsu Han. Accurate latency-based congestion feedback for datacenters. In *USENIX Annual Technical Conference*, pages 403–415, 2015.

32 Zhi Li, Anthony Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. *Netflix Tech Blog*, 2016. URL: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652.

33 Cicco Luca De Gaetano Carlucci, and Saverio Mascolo. Experimental investigation of the google congestion control for real-time flows. In *Proceedings of the ACM SIGCOMM 2013 Workshop on Network and System Support for Games (NetGames 13)*, page 2126, 2013. doi:10.1145/2491172.2491182.

34 Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.

35 Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 193206, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544225.

36 Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

37 Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using fec for conversational multimedia communication. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 191–202, 2014.

38 Netflix. VMAF - Video Multi-Method Assessment Fusion. `https://github.com/Netflix/vmaf`, 2016.

39 Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Usenix annual technical conference*, pages 417–429, 2015.

40 Abhishek Prasad, Anh Le, Sanjay Jha, and Richard Gibbens. itelescope: Intelligent video telemetry and classification in real-time using software defined networking. In *2018 4th IEEE Conference on Network Softwarization (NetSoft)*, pages 401–409. IEEE, 2018. `doi:10.1109/NETSOFT.2018.8460000`.

41 Manish Ravinet. Mahimahi Traces. `https://github.com/ravinet/mahimahi/tree/master/traces`, 2016.

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

43 Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

44 Sea Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. Low extra delay background transport (ledbat). Technical report, 2012.

45 Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions On Networking*, 28(4):1698–1711, 2020.

46 Shibo Wang, Shusen Yang, Xiao Kong, Chenglei Wu, Longwei Jiang, Chenren Xu, Cong Zhao, Xuesong Yang, Jianjun Xiao, Xin Liu, Changxi Zheng, Jing Wang, and Honghao Liu. Pudica: Toward Near-Zero queuing delay in congestion control for cloud gaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 113–129, Santa Clara, CA, April 2024. USENIX Association. URL: `https://www.usenix.org/conference/nsdi24/presentation/wang-shibo`.

47 Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

48 WebRTC. `https://webrtc.org/`.

49 WebRTC Developers. WebRTC Congestion Control Probe Controller Source Code. `https://chromium.googlesource.com/external/webrtc/+/3c1e558449309be965815e1bf/webrtc/modules/congestion_controller/probe_controller.cc`, 2023.

50 Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 459–471, 2013.

51 Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Alexander Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *NSDI*, volume 20, pages 495–511, 2020.

52 Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018.

53 Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

54 YouTube. Dynamic video. `https://www.youtube.com/watch?v=19ikl8vy4zs`, 2023.

55 Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 509–522, 2015.

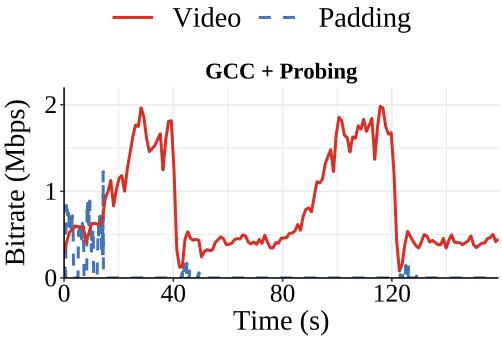
56 Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 775788, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3447993.3483259.

57 Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

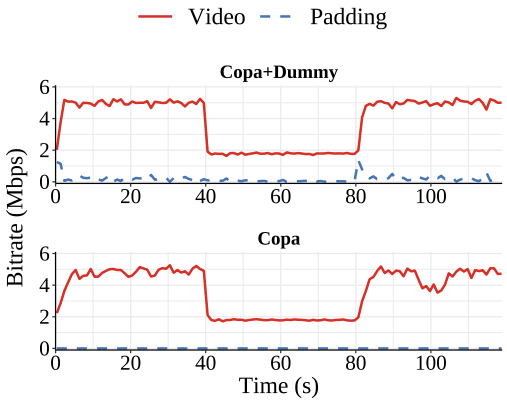
58 Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3300061.3345430.

59 Chao Zhou, Wenjun Wu, Dan Yang, Tianchi Huang, Liang Guo, and Bing Yu. Deadline and priority-aware congestion control for delay-sensitive multimedia streaming. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 4740–4744, 2021.

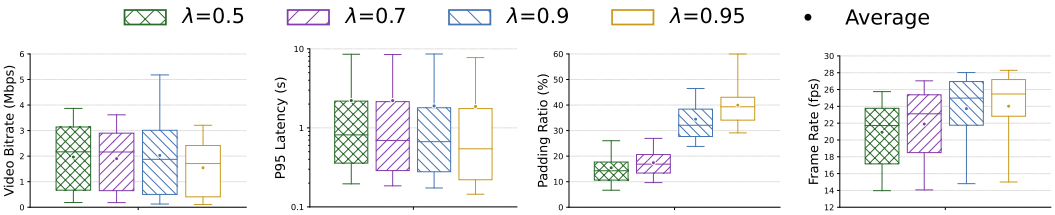
60 Xiaoqing Zhu, Rui Pan, Marcello Ramalho, Sumit Jones, Yutaka Tokuda, Raja Subramanian, and Philip Zhu. Nada: A unified congestion control scheme for real-time media. In *Proceedings of the IEEE Packet Video Workshop*, 2013.



(a) Ad-hoc probing behavior in WebRTC on a periodic link alternating between 2 Mbps and 500 Kbps every 40s. Padding traffic is used when GCC's estimate drops sharply (t=40s), but not when capacity increases (t=80s).



(b) Copa with dummy traffic converges faster to available capacity and maintains a smoother steady-state video bitrate.



(a) Video Bitrate **(b)** Log P95 Latency **(c)** Padding Ratio **(d)** Frame Rate

Figure 22 Effect of λ on Vidaptive's performance. Increasing the value of λ increases the frame rate and padding ratio and decreases the video bitrate and latency on fluctuating links.

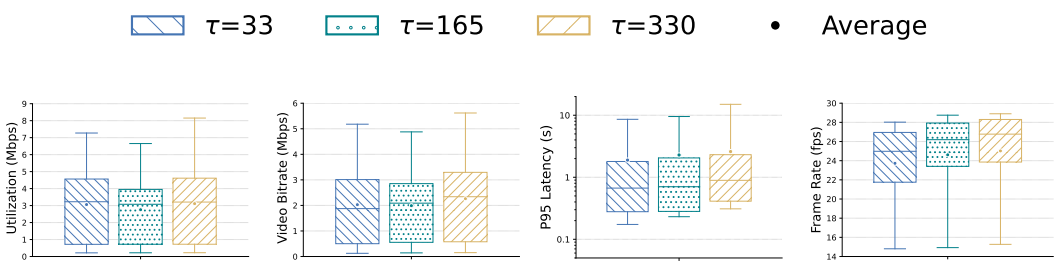
A Probing Mechanisms

Fig. 21a displays the performance of probing mechanism within GCC [49]. This probing controller typically sends probes (padding) in three scenarios: 1) When initiating a connection or after a significant idle period. 2) Upon detecting a potential increase in available bandwidth. 3) Periodically, as part of maintaining an up-to-date estimate of the network conditions, especially when dynamic changes in the network are suspected. We tested this probing mechanism within Google's WebRTC on a link alternating between 2 Mbps and 500 Kbps every 40 seconds. The padding traffic is only sent when GCC significantly reduces the video bitrate (t=40s), but it does not help with bandwidth discovery (t=80s) when the link rate increases. This approach to sending the padding traffic has essentially not provided much benefit for GCC.

B Extended Evaluation

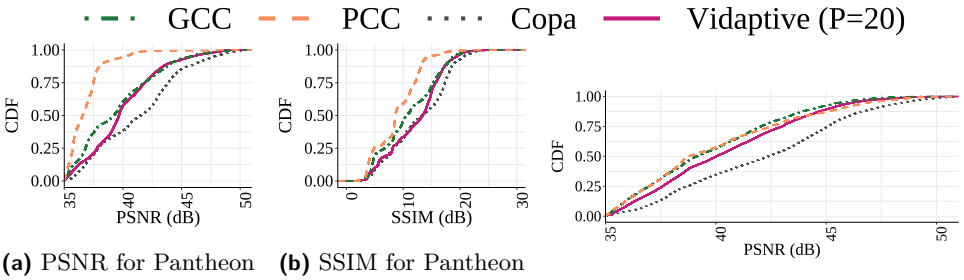
More Metrics. Fig. 24 shows the other metrics for the experiment in the paper. The trends are consistent. Vidaptive has better video quality compared to GCC and PCC.

Effect of λ. We evaluate the impact of the parameter λ, which dictates what fraction of the frame service times we want to tightly bound on all traces and 5 videos. Fig. 22 shows the distribution of the metrics across all the videos and traces with $\lambda = 0.5, 0.7, 0.9, 0.95$. When λ increases, the bitrate selection in §3.3 is stricter on the frame service time values



(a) Link Utilization **(b)** Video Bitrate **(c)** Log P95 Latency **(d)** Frame Rate

Figure 23 Effect of pacer queue pause threshold (τ) on Vidaptive. As τ increases, the frame rate increases because fewer frames are skipped, consequently, P95 latency increases. The video bitrate does not change much since the encoder bitrate selector keeps the frame service time under control.



(a) PSNR for Pantheon traces **(b)** SSIM for Pantheon traces **(c)** PSNR for Cellular traces

Figure 24 CDF across all the frames.

and picks more conservative ones. Hence, the video bitrate decreases (Fig. 22a), and the latency decreases as well (Fig. 22b). Since Vidaptive uses dummy traffic, changes in the video bitrate do not affect CCA estimations and consequently do not change the overall link utilization. Therefore, increasing λ increases the padding ratio. The average frame rate increases (Fig. 22d) as more frames are permitted since their latency is below the threshold.

Pacer Queue Pause Threshold (τ). Fig. 23 shows how the pacer queue pause threshold, τ (§3.2), affects the performance of Vidaptive. Vidaptive skips the current frame if the previous frame has been more than τ in the pacer queue. We tested Vidaptive with τ = 33, 165, 330 ms. Changing τ does not change the network utilization (Fig. 23a) because dummy traffic decouples congestion control from the encoder, padding any encoder output to match the link rate. The encoder bitrate selection logic computes the target bitrate based on the actual frame service time values, so changing τ does not affect the video bitrate much (Fig. 23b) as frame service time does not count for the queueing delay frames experience before they're sent. As τ increases, the frame rate increases (Fig. 23d) because fewer frames are skipped, causing an increase in the P95 latency (Fig. 23c).

Vidaptive selects $\tau = 33ms$ as it has low latency and relatively high frame rate compared to GCC.

C Example of Effect of Dummy Traffic

To quantify the effect of dummy traffic in more detail, we disable the changes we made to the target bitrate selection logic and focus on transport layer changes (§3.1). In Fig. 21b, we emulate a link that starts with 5 Mbps of bandwidth for 40 s, drops to 2 Mbps for the

Video URL	Category	Avg. Bitrate (Kbps)
Video 1	Ice Skating	4127
Video 2	Wild Animals Collection	4883
Video 3	High-motion Cooking	7278
Video 4	YouTube Vlogger	3025
Video 5	Animated Movie	2352
Video 6	Videocall Stock Footage	1751
Video 7	Press Conference	1410
Video 8	YouTube Vlogger	1152
Video 9	Music Video	4227
Video 10	Music Video	4144
Video 11	YouTube Vlogger	2199
Video 12	YouTube Vlogger	1379
Video 13	Art	3636
Video 14	Low-motion Cooking	2502
Video 15	Football	6256
Video 16	Crafts	2532
Video 17	Press Conference	1847
Video 18	Screen Sharing / Education	40
Video 19	Live Music	4184
Video 20	Weather Forecast	4754

Table 1 Details of our dataset. All videos are at 1920×1080 at 30 FPS and are 4 min long.

next 40 s before jumping back to 5 Mbps. We compare the video and padding bitrate for "Copa" to Copa with dummy traffic ("Copa+Dummy"). Copa takes 6 s to match the link rate, while Copa+Dummy takes 2 s. The dummy traffic is sent only when the video traffic cannot match the link capacity when it suddenly opens up (around 0s and 80s). Copa does not match capacity as fast because its rate on the wire is determined by the slow-reacting encoder (§2). Further, "Copa+Dummy" has a more stable steady-state bitrate than "Copa" because the dummy traffic decouples the CCA's feedback from the video encoder's variable output, enabling more accurate link capacity estimation.

D Videos

Dataset Information Tab. 1 summarizes the information of all the videos that we used in the experiments. The videos are all collected from YouTube and cover a different range of motions and settings.