

SQP: Congestion Control for Low-Latency Interactive Video Streaming

Devdeep Ray

Carnegie Mellon University, Google

Connor Smith

Google

Teng Wei

Google

David Chu

Google

Srinivasan Seshan

Google

Abstract

This paper presents the design and evaluation of SQP¹, a congestion control algorithm (CCA) for interactive video streaming applications that need to stream high-bitrate compressed video with very low end-to-end frame delay (e.g. AR streaming, cloud gaming). SQP uses frame-coupled, paced packet trains to sample the network bandwidth, and uses an adaptive one-way delay measurement to recover from queuing, for low, bounded queuing delay. SQP rapidly adapts to changes in the link bandwidth, ensuring high utilization and low frame delay, and also achieves competitive bandwidth shares when competing with queue-building flows within an acceptable delay envelope. SQP has good fairness properties, and works well on links with shallow buffers.

In real-world A/B testing of SQP against Copa in Google’s AR streaming platform, SQP achieves 27% and 15% more sessions with high bitrate and low frame delay for LTE and Wi-Fi, respectively. When competing with queue-building traffic like Cubic and BBR, SQP achieves 2 – 3× higher bandwidth compared to GoogCC (WebRTC), Sprout, and PCC-Vivace, and comparable performance to Copa (with mode switching).

1 Introduction

In recent years, there has been an increasing interest in deploying a new class of video streaming applications: low-latency, interactive video streaming [1]. These applications offload demanding graphics-intensive workloads like video games and 3D model rendering to powerful cloud servers at the edge, and stream the application viewport to clients over the Internet in the form of compressed video. Examples of deployed applications that use this approach include cloud gaming services (e.g., Amazon Luna [2], Google Stadia [3], Microsoft xCloud [4], and NVIDIA GeForce Now [5]), remote desktop services (e.g., Azure Virtual Desktop [6], Chrome Remote Desktop [7], and others), and cloud augmented reality services (e.g., 3D car search on Android [8], ARR [9], NVIDIA CloudXR [10],) that enable high quality augmented reality (AR) on mobile devices.

For the end user experience to be comparable to running the applications locally, the video must be encoded at the highest bitrate that still allows the frames to be transmitted and received with

minimal delay. A CCA for low-latency interactive video streaming must have the following properties:

- (1) Low Queuing Delay: The CCA must be able to probe for more bandwidth without causing excessive queuing, and must quickly back off when the available bandwidth decreases in order to reduce in-network queuing. CCAs like Cubic [11] fill up network queues until packet loss occurs, and some CCAs, like PCC [12], are slow to react to drops in bandwidth, resulting in very high delays that are unacceptable for low-latency interactive streaming.
- (2) Fairness: The CCA must achieve high, stable bandwidth when competing with queue-building flows (e.g., Cubic, BBR [13]), while achieving low delay when running in isolation. Some low-delay CCAs have explicit mechanisms to prioritize throughput over delay when queue-building cross traffic is detected, but they can be inherently unstable (e.g., Copa [14] can misdetect self-induced queuing as competing traffic, resulting in additional self-induced queuing [15]), while others are slow to converge (e.g., Nimbus [15] operates over 10s of seconds). In addition, multiple homogeneous flows must also converge to fairness quickly.
- (3) Friendliness: The CCA must be friendly to other CCAs and must avoid starving them - GoogCC’s [16–18] adaptive threshold mechanism for competing with cross traffic can occasionally starve other flows (e.g., Cubic).
- (4) Video Awareness: The CCA must accommodate encoder frame size variations, and achieve bandwidth probing in application-limited scenarios without the need for frame padding. The CCA must use a rate-based congestion control mechanism to minimize the end-to-end frame delay - the bursty nature and time-varying throughput of window-based mechanisms necessitate an undesirable trade-off between bandwidth utilization, encoder rate-control updates, and the end-to-end frame delay.

While most CCAs aim to achieve high throughput, low delay, and competitive performance when competing with queue-building flows, simultaneously achieving these requirements is challenging in an environment as diverse as the Internet. Choosing the right trade-offs and correctly prioritizing the design requirements (listed above in decreasing order of priority) enables a design that is highly optimized for the specific application class. Existing CCAs make different trade-offs based on their particular design goals, and some of these design choices make them unsuitable for low-latency interactive streaming applications.

¹Streaming Quality Protocol, Snoqualmie Pass



This work is licensed under a Creative Commons Attribution International 4.0 License.

In this paper, we present SQP, a novel congestion control algorithm that was developed in conjunction with Google’s AR streaming platform. SQP’s key features are listed below:

- (1) Prioritizing Delay over Link Utilization: Since delay is more critical for the QoE of low-latency interactive video streaming applications, SQP sacrifices peak bandwidth utilization when running in isolation in order to achieve low delay and delay stability. For example, on a 20 Mbps link where SQP is the only flow, it is acceptable to utilize 18 Mbps if this trade-off reduces delays across a wider range of scenarios.
- (2) Application-specific Trade-offs : SQP is designed for low-latency interactive streaming applications, which have specific requirements in terms of minimum bandwidth and maximum delay. If these parameters are outside the acceptable range due to external factors (e.g., poor link conditions, very high delays due to queue-building cross traffic), it is acceptable to end the streaming session. In contrast to traditional algorithms, SQP restricts its operating environment, which enables SQP to achieve acceptable throughput and delay performance across a wider range of relevant scenarios.
- (3) Frame-focused Operation : In-network queuing is a key mechanism that allows CCAs to detect the network capacity. CCAs that probe infrequently (e.g., PCC, GoogCC) have lower average delay, but suffer from link underutilization on variable links. SQP piggy-backs bandwidth measurements onto each frame’s transmission by sending each frame as a short (paced) burst, and updates its bandwidth estimate after receiving feedback for each frame. For low-latency interactive streaming applications, the QoE is determined by the end-to-end frame delay, and not just the in-network queuing delay. SQP network probing relies on queuing at the sub-frame level without increasing the end-to-end frame delay, and is able to adapt to changes in network bandwidth much faster than protocols like PCC [12, 19] and GoogCC [17].
- (4) Direct Video Bitrate Control : SQP uses frame-level bitrate changes in order to respond to congestion, and drains self-induced queues by reducing the video bitrate. SQP’s rate-based congestion control minimizes the end-to-end frame delay compared to protocols that are window-based (Copa), or throttle transmissions for network RTT measurements (BBR).
- (5) Competitive Throughput : SQP’s bandwidth probing and sampling mechanism is competitive by default, and achieves high, stable throughput share when competing with queue-building flows that cause delays within an acceptable range. SQP avoids high queuing delays and starving other flows using mechanisms like adaptive one-way delay measurements, a bandwidth target multiplier, and frame pacing. SQP’s design avoids the pitfalls of delay-based CCAs that use explicit mode switching (e.g., Copa [14, 15]).

SQP’s evaluation on real-world wireless networks for Google’s AR streaming platform, and across a variety of emulated scenarios, including real-world Wi-Fi and LTE traces show that:

- (1) Under A/B testing of SQP and Copa² on Google’s AR streaming platform across ≈ 8000 individual streaming sessions,

²Adapted from mvfst [20], does not implement Copa’s mode switching.

71% of SQP sessions on Wi-Fi have P50 bitrate > 3 Mbps and P90 frame RTT < 100 ms, compared to 56% for Copa. On cellular links, 36% of SQP sessions meet the criteria versus only 9% for Copa.

- (2) Across emulated wireless traces, SQP’s throughput is 11% higher than Copa (without mode switching) with comparable P90 frame delays, while Copa (with mode switching), Sprout [21], and BBR incur a 140–290% increase in the end-to-end frame delay relative to SQP.
- (3) SQP achieves high and stable throughput when competing with buffer-filling cross traffic. Compared to Copa (with mode switching), SQP achieves 70% higher P10 bitrate when competing with Cubic, and 36% higher link share when competing with BBR.

This work does not raise any ethical issues.

2 Related Work

A suitable congestion control algorithm for low-latency interactive video streaming must satisfy the four key properties discussed in § 1. Various CCAs are summarized in Table 1. Low-latency CCAs like TCP-Lola [27], TCP-Vegas [28], and Sprout (Salsify³) [21, 29] that use packet delay as a signal have a key drawback: they are unable to achieve high throughput when competing with queue-building cross-traffic. Some mode switching algorithms (e.g., Copa [14]) can misinterpret self-induced queues as competing flows, resulting in high delays, whereas other CCAs like Nimbus [15], and GoogCC (WebRTC) [17, 18] converge slowly, and can have unstable throughput when competing with queue-building flows.

BBR [13] periodically throttles transmissions to measure a baseline delay, which is problematic for interactive video streaming since frames cannot be transmitted during its minRTT probe. Window-based protocols have a similar problem - they transmit packets in bursts, and a mismatch between packet transmissions and frame generation require sender-side buffering, and increase the end-to-end frame delay.

Utility-based algorithms like PCC [12, 19] explicitly probe the network and aim to maximize a utility function based on throughput, delay, and packet loss. These CCAs converge slowly on dynamic links, and have inconsistent performance when competing with queue-building flows.

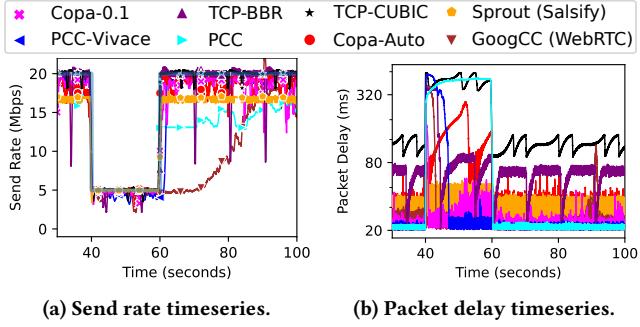
CCAs use rate-based or window-based mechanisms in order to control the transmission rate under congestion. Rate-based CCAs are better suited for video streaming due to the burst-free nature of packet transmissions, whereas the bursty window-based mechanisms can block frame transmissions at the sender and make encoder rate-control challenging (e.g., Salsify-Sprout). The other benefit of rate-based CCAs is that their internal bandwidth estimate can be used to directly set the video bitrate, whereas window-based mechanisms require additional mechanisms for setting the video bitrate.

3 Preliminary Study

To illustrate the shortcomings of existing congestion control algorithms in the context of low-latency interactive streaming,

³Salsify streamer uses Sprout as the CCA (used interchangeably)

Protocol Category	Congestion Detection	Competing with Queue-building Flows	Congestion Control Mechanism	Comments
Explicit signaling DCTCP, ABC, XCP [22–24]	Explicit signals from network to detect congestion	Compete with homogeneous flows	Various	Lack of support, traffic heterogeneity - unsuitable for Internet-based interactive video streaming
Low Delay TCP-Lola, TCP-Vegas, Sprout (Salsify), TIMELY, Swift [25, 26]	Packet delay/delay-gradient, stochastic throughput forecast (Sprout)	Queue-building flows cause low throughput	Window-based, Rate-based (TIMELY), hybrid (Swift)	High, stable throughput required - not achieved with queue-building cross-traffic, custom encoder for handling bursty CCA (Salsify)
Mode Switching Copa, Nimbus, GoogCC (WebRTC),	Packet delay/delay-gradient as congestion signal	More aggressive when competing flow detected	Window-based, Rate-based (GoogCC)	Mode-switching is unstable (Copa, GoogCC), can be slow to converge (Nimbus, GoogCC)
Model-Based BBR	minRTT probe, pacing gain for bandwidth	Designed to be competitive with Cubic	Rate- and window-based	200ms minRTT probe throttles transmissions, 2 BDP in-flight under ACK aggregation/competition
Utility-based PCC, PCC-Vivace	Explicit probing, delay, packet loss	Measure network response to rate change	Rate-based	Inconsistent performance with queue-building flows, slow convergence on dynamic links

Table 1: Various CCAs that exist today, and their properties.**Figure 1: Congestion control performance on a variable link (link bandwidth shown as a shaded light blue line).**

we present some preliminary experimental results using the Pantheon [30, 31] testbed. Details regarding the specific CCA implementations are provided in § 6.1.

3.1 Variable Bandwidth Link

We ran a single flow for 120 seconds over a 40 ms RTT link, where the bandwidth temporarily drops down to 5 Mbps from 20 Mbps. The goal of this experiment is to see if the algorithm can (1) quickly discover additional bandwidth when the available bandwidth increases, and (2) maintain low delay when the available bandwidth decreases.

Figure 1 shows the throughput and delay timeseries between $t = 30$ s and $t = 100$ s. Throughout the entire trace, Cubic operates with the queues completely full, resulting in high link utilization at the cost of high queuing delays. Among the low-delay algorithms, the delay performance differs greatly across specific algorithms. When the link rate goes down to 5 Mbps, Sprout and COPA-0.1 (Copa without mode switching, $\delta = 0.1^4$) are able to adapt rapidly without causing a delay spike. PCC-Vivace [19], GoogCC and BBR are slower to adapt, causing 3-8 seconds long delay spike. Throughout the low bandwidth period, PCC maintains persistent, high queuing

⁴A lower delta makes COPA more aggressive, sacrificing low delay for higher throughput. The original paper proposes using 0.5, whereas the version on Pantheon uses 0.1. Facebook's testing of COPA [32] used 0.04.

delay, whereas COPA-Auto (Copa with mode-switching, adaptive δ) incorrectly switches to competitive mode, significantly increasing queuing delay.

In addition to the delay that occurs when the link rate drops, some algorithms have inherently more queuing than others. BBR can maintain up to 2 BDP in-flight, causing up to 1 BDP of in-network queuing. Both, COPA-0.1 and COPA-Auto demonstrate significant short-term delay variations due to COPA's 5-RTT probing cycle, which serves the role of probing the network for additional capacity. The peak delay is inversely proportional to the value of δ , and is worse in the case of COPA-Auto, since it periodically misinterprets its own delay as delay caused due to a competing queue-building flow, and consequently reduces the value of δ in response. There is significant variation in Sprout's packet delay due to the bursty nature of its packet transmissions, even though it is significantly underutilizing the link. GoogCC also demonstrates a delay spike around $t = 90$ s, when its send rate hits the link limit after an extended ramp-up period. PCC-Vivace, COPA-Auto, COPA-0.1, and BBR are able to rapidly probe for more bandwidth when the link rate increases. On the other hand, PCC and GoogCC are the slowest to converge, taking more than 20-30 seconds to ramp up after the link rate increases, resulting in severe underutilization.

In order to achieve high link utilization and low delays for low-latency interactive video streaming, the CCA must quickly identify the link capacity without causing queuing delays, and quickly back off when the delay is self-induced. SQP is able to achieve these requirements, as shown in Section 6.3.

3.2 Short Timescale Variations

In this section, we examine the short timescale behavior of existing protocols to see if they can provide the low packet delay and stable throughput [33] needed to support the requirements of low-latency streaming applications. We present three algorithms that demonstrate distinct short-term behavior: COPA-Auto, BBR and Vivace (additional results in Section 6.7). COPA-0.1 and Sprout behave similar to COPA-Auto, and PCC behaves similar to Vivace in these experiments. We ran each algorithm on a fixed 20 Mbps link with 20 ms of delay in each direction. Figure 2a shows the one-way

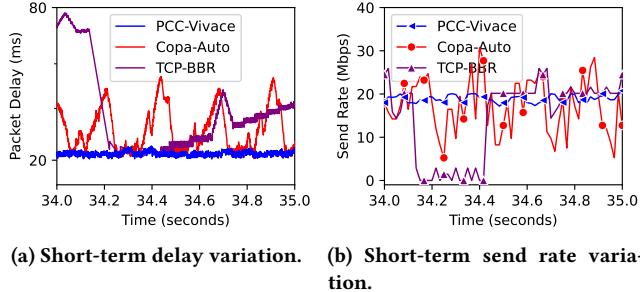


Figure 2: A closer look at the short term delay and send rate variation on a constant 20 Mbps link.

packet delays and Figure 2b shows the packet transmission rate for each frame period (16.66 ms at 60FPS).

Copa-Auto’s one-way packet delay oscillates between 20 ms and 60 ms over a 12-frame period, with large variations in the send rate at frame-level timescales. If a smooth video bitrate is determined using the average send rate to maximize utilization, the frames at $T = 34.3, 34.5$ would get delayed at the sender. To lower the sender-side queuing delay, the encoder rate selection mechanism must either: (1) choose a conservative video bitrate, resulting in underutilization, or (2) have frequent rate control updates.

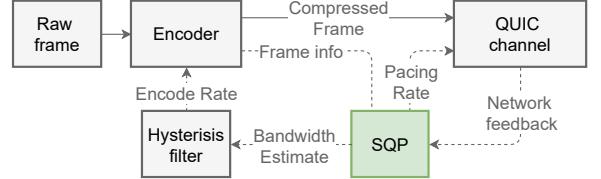
While BBR is not particularly suitable for interactive streaming because of its higher queuing delay, BBR’s RTT probing mechanism is especially problematic. Every 10 seconds, BBR throttles transmissions (transmitting at most 4 packets per round trip) for 200 ms to measure changes in the link RTT (between $T = 34.2$ and $T = 34.4$). During this period the generated video frames will be queued at the sender, resulting in 200 ms of video stutter every 10 seconds.

Rate-based algorithms like PCC and Vivace are better suited for streaming applications, since the internal rate-tracking mechanism can be used to set the video bitrate, and frames are not delayed at the sender if the encoded frames do not overshoot the requested target bitrate. While Salsify [29] attempts to solve this problem using a custom encoder that can match the instantaneous transmission rate of a bursty CCA like Sprout [21], rearchitecting the CCA is a more universal solution that can leverage advances in hardware video codecs that have good rate control mechanisms (App. A).

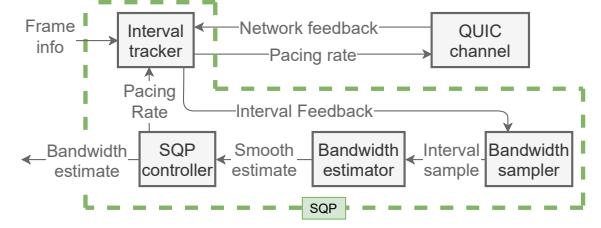
To minimize the end-to-end frame delay, the CCA must transmit encoded frames immediately, and pace faster than the rate at which the network can deliver the packets. SQP directly controls the video bitrate using smooth bandwidth estimates, and the transmissions are synchronized with the frames, which reduces the end-to-end frame delay (Section 6.7).

4 Design

Low-latency interactive streaming applications generate raw frames at a fixed frame-rate. The video bitrate is determined by an adaptive bitrate (ABR) algorithm using signals from the CCA in order to manage frame delay, network congestion, and bandwidth utilization. The compressed frames are transmitted over the network, and eventually decoded and displayed at the client device.



(a) Video streaming architecture.



(b) SQP internal architecture.

Figure 3: Low-latency video streaming and SQP architectures.

SQP is a rate- and delay-based CCA for low-latency interactive video streaming, and aims to (1) provide real-time bandwidth estimates that ensure high utilization and low end-to-end frame delay on highly variable links, and (2) achieve competitive throughput in the presence of queue-building cross traffic. SQP’s congestion control mechanism must be purely rate-based in order to avoid the undesirable trade-offs between bandwidth utilization, encoder bitrate changes, and the end-to-end frame delay (§ 1).

4.1 Architecture Overview

SQP’s role in the end-to-end streaming architecture and its key components are shown in Figures 3a and 3b. SQP relies on QUIC [34] to reliably transmit video frames, perform frame pacing, and provide packet timestamps for estimating the network bandwidth. SQP directly controls the video bitrate, and a simple hysteresis filter serves as a bridge between SQP and the encoder to reduce the frequency of bitrate changes.

Internally, SQP’s components work together in order to achieve the key design goals:

- (1) Bandwidth Probing: SQP transmits each frame as a short, paced burst, and the bandwidth sampler uses frame-level packet dispersion statistics from the interval tracker for discovering additional bandwidth.
- (2) Recovery from Transient Queues: SQP’s bandwidth samples are penalized when the delay increases over a short period (§ 4.2), enabling it to recover from transient self-induced queuing.
- (3) Recovery from Standing Queues: SQP uses a target multiplier mechanism (§ 4.5) to maintain some slack in the link utilization, enabling recovery from self-induced standing queues. SQP remains competitive when competing flows cause standing queues (within acceptable delay limits) since it uses a small, dynamic window to track the transient delay (§ 4.3).
- (4) Rate-based congestion control: SQP aims to carefully pace each frame faster than the rate at which the network can

- deliver the packets (§ 4.5), and responds to congestion by smoothly changing the bandwidth estimate (and consequently, the video bitrate) using gradient-based updates (§ 4.4). As opposed to using ACK-clocking and window-based mechanisms, rate-based congestion control simplifies integration with the video encoder and reduces the end-to-end frame delay (§ 3.2).
- (5) **Fairness and Interoperability:** SQP’s bandwidth estimator (§ 4.4) is based on maximizing a logarithmic utility function, which improves dynamic fairness due to its AIMD-style updates (§ 5.2). SQP’s frame pacing and the bandwidth target multiplier mechanisms ensure dynamic fairness across multiple SQP flows (§ 5.2), and provide a theoretical upper bound on SQP’s share when competing with elastic flows (§ 5.1).

For the initial part of the discussion, we will assume the existence of a ‘perfect’ encoder with the following properties: (1) the target bitrate can be changed on a per-frame basis without any negative consequences, as long as the target bitrate does not change significantly from frame to frame, and (2) the encoder does not overshoot or undershoot the specified target rate. In § 6.10, we discuss how SQP works in a practical setting when encoders do not satisfy some of these assumptions.

4.2 Bandwidth Sampling

The goal of SQP’s bandwidth sampling algorithm is to measure the end-to-end frame transport rate that achieves high link utilization while avoiding self-induced queuing and packet transmission pauses (e.g., Copa and BBR in § 3.2). SQP transmits each frame as a short burst that is faster than the network delivery rate, which causes a small amount of queuing. This queue is drained by the time the next frame arrives at the bottleneck if the average video bitrate is lower than the available bottleneck link capacity. SQP uses the dispersion of the frame-based packet train [35] to measure link capacity, with some key differences that aid in congestion control compared to basic packet-train techniques. SQP’s probing works at sub-frame timescales (< 16.66 ms @ 60FPS), in contrast to CCAs that probe for bandwidth over longer timescales (PCC:2RTT, BBR:1 min-RTT, and Copa:2.5 RTT).

Consider an application generating frames at a fixed frame rate. As shown in Figure 4, a frame of size F is transmitted every inter-frame time interval, I (e.g., 16.66 ms at 60FPS), and the average bitrate is $\frac{F}{I}$. Each frame is paced at a rate that is higher than $\frac{F}{I}$ (shown by the steep slope of the green dots). If there are no competing flows (competition scenario discussed in § 5.1), and the link bandwidth is lower than the pacing rate, the packets will get spaced out according to the bandwidth of the bottleneck link (slope of the red dots). SQP computes the end-to-end frame transport bandwidth sample as:

$$S = \frac{F}{R_{end} - S_{start} - \Delta_{min}} \quad (1)$$

This is the slope of the red dotted line in Figure 4. S_{start} and R_{end} are the send and receive times of the first and last packets of a frame, respectively, and Δ_{min} is the minimum one-way delay (delta between send and receive timestamps) for packets sent during a small window in the past (§ 4.3). Δ_{min} and $R_{end} - S_{start}$ have the

same clock synchronization error (sender-side vs. receiver-side timestamps) and cancel each other out.

Underutilization Sample: When the network is underutilized or 100% utilized, no additional queuing occurs across multiple frames ($\Delta = R_{start} - S_{start} = \Delta_{min}$ remains constant). Thus, as shown in Figure 4a, the sample is equal to the packet receive rate for a frame (i.e. the bottleneck link bandwidth). The samples during link underutilization are higher than the video bitrate ($\frac{F}{I}$), and SQP increases its bandwidth estimate. When the link is 100% utilized, the samples are equal to the video bitrate, indicating good link utilization.

Overutilization Sample: Transient overutilization due to frame size overshoots, bandwidth overestimation (link aggregation, token bucket policing), or a drop in network bandwidth can cause queuing that builds up across multiple frames. This results in an increase in $\Delta - \Delta_{min}$ for subsequent frames, which lowers the bandwidth samples for subsequent frames (Figure 4b, slope of dotted red line for the second frame). Thus, SQP lowers the video bitrate below the link rate and recovers from transient queuing. When packets are lost, SQP scales down its samples by the fraction of lost packets, primarily responding to sustained loss events (e.g., shallow buffers, § 6.6).

Video Encoder Undershoot: While SQP is also able to discover the link bandwidth quickly in application-limited scenarios since it relies on the pacing burst rate, and not the average video bitrate, bandwidth samples from small frames are unfairly penalized due to delay variations. SQP often has to deal with application-limited scenarios where the bitrate of the encoded video is less than the bandwidth estimate. This can be due to conservative rate control mechanisms that serve as a bridge between the bandwidth estimate and the encoder bitrate to reduce the frequency of encoder bitrate updates, or due to encoder undershoot during low complexity scenes that do not warrant encoding frames at the full requested target bitrate (e.g. low-motion scenes like menus). When SQP is application-limited, queuing delay from past frames can unfairly penalize the bandwidth sample (Figure 4c). While padding bytes can be used to bring up the video bitrate to SQP’s bandwidth estimate, this results in wastage of bandwidth. To improve SQP’s robustness under application-limited scenarios, we modify the bandwidth sampling equation to account for undershoot:

$$S = \frac{F \cdot \gamma}{R_{end} - S_{start} - \Delta_{min} + (R_{end} - R_{start}) \cdot (\gamma - 1)} \quad (2)$$

where $\gamma = \frac{F_{max}}{F}$ is the undershoot correction factor, F_{max} is the hypothetical frame size without undershoot, and $(R_{end} - R_{start}) \cdot (\gamma - 1)$ is the predicted additional time required for delivering the hypothetical full-sized frame. This computes the bandwidth sample by extrapolating the delivery of a small frame to the full frame size that is derived from SQP’s current estimate. In Figure 4c, the solid dots are actual packets for a frame, and the hollow dots show the extrapolated transmission and delivery of the packets.

4.3 Tracking Minimum One-way Delay (Δ_{min})

The minimum packet transmission delay, Δ_{min} , serves as a baseline for detecting self-inflicted network queuing. The window size for tracking Δ_{min} represents the duration of SQP’s memory of Δ_{min} ,

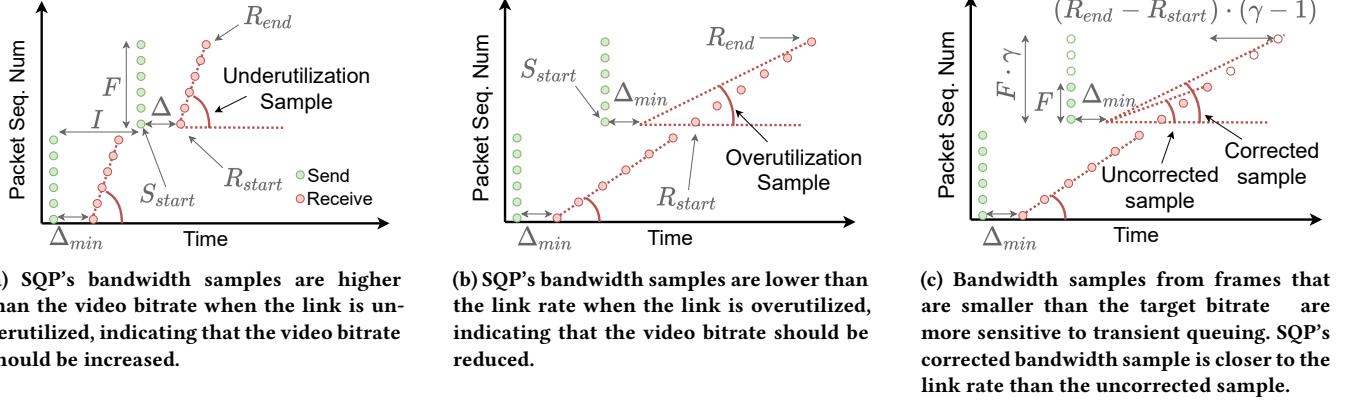


Figure 4: SQP's bandwidth samples converge towards the link rate and aid in draining self-inflicted queues. The slope of the dotted red line represents the bandwidth sample in each case.

which affects SQP's self-induced queuing and throughput when competing with queue-building cross traffic. If a small, fixed window were used (e.g., 0.1-0.5 s), when self-induced queuing occurs, Δ_{min} could expire before SQP can recover. While a larger, fixed window (e.g., 10-30 s) would aid recovery from self-induced queues by anchoring SQP to the lowest one-way delay observed over the window, SQP's bandwidth samples would be more sensitive to delay variations caused by the queue-building cross traffic, lowering SQP's throughput share.

To balance these trade-offs, SQP uses an adaptive window size of $2 \times sRTT$ [36]. This has two advantages. First, for self-induced queuing, SQP's adaptive window grows quickly, and in conjunction with SQP's bandwidth target multiplier mechanism (§ 4.5), enables SQP to drain self-induced queues. Second, when competing queue-building flows build standing queues, Δ_{min} quickly increases in response, so that SQP doesn't react to the competitor's standing queue. Since SQP paces frames into the network faster than SQP's current share, it can probe for more bandwidth when competing with queue-building flows, even if the combined link utilization of SQP and the cross traffic is near 100%. Together, this enables SQP to obtain a high throughput share when those queue-building flows (1) do not cause very high delays, and (2) have low queuing delay variation over periods of $2 \times sRTT$ (§ 6.5).

While the role of SQP's Δ_{min} mechanism is similar to BBR's min-RTT mechanism, SQP does not need an explicit probing mechanism for Δ_{min} since it (1) increases the window size when self-induced queuing occurs, and (2) reduces the video bitrate to drain the self-induced queue, which provides organic stability. While BBR's explicit probing of the baseline network RTT is more accurate, the need to significantly limit packet transmissions for 200 ms makes this approach unsuitable for real-time interactive streaming media. We evaluate the impact of the window size scaling parameter in § 5.3.

4.4 Bandwidth Estimate Update Rule

SQP's bandwidth estimator processes noisy bandwidth samples measured by SQP's bandwidth sampler to provide a smooth bandwidth, which is used to set the video encoder bitrate. SQP's update

rule is inspired from past work on network utility optimization [37], and is derived by optimizing a logarithmic reward for higher bandwidth estimates and a quadratic penalty for overestimating the bandwidth:

$$\max \log(1 + \alpha \cdot B) - \beta \cdot (B - e)^2 \quad (3)$$

where B is SQP's bandwidth estimate, α is the reward weight for a higher bandwidth estimate, β is the penalty for overestimating the bandwidth, and e is a parameter derived from the bandwidth sample S , such that the function is maximized when $B = S$. Taking the derivative of this expression and evaluating the expression with B set to the current estimate provides a gradient step towards the maxima. Simplifying the derivative of the expression 3, and the constant expressions involving α and β , the update rule can be rewritten as

$$B' = B + \delta \left(r \left(\frac{S}{B} - 1 \right) - \left(\frac{B}{S} - 1 \right) \right) \quad (4)$$

B' and B are the updated and current estimates, r is the reward weight for bandwidth utilization and δ is the step size and represents a trade-off between the smoothness of the bandwidth estimate and the convergence time under dynamic network conditions. SQP empirically sets $\delta = 320$ kbps, and $r = 0.25$.

SQP's target and pacing multiplier mechanisms (§ 5.1) work in conjunction with the update rule to improve SQP's convergence to fairness (§ 6).

4.5 Pacing and Target Multipliers

SQP's design includes two key mechanisms for ensuring friendliness with other flows - instead of transmitting each frame as an uncontrolled burst at line rate, SQP paces each frame at a multiple of the bandwidth estimate, and targets a slightly lower video bitrate than the samples (determined by a target multiplier). Suppose SQP is sharing a bottleneck link with a hypothetical elastic CCA [15] that perfectly saturates the bottleneck link without inducing any queuing delay. If SQP transmitted frames as uncontrolled bursts, the elastic flow might not be able to insert any packets between SQP's packets. Thus, the bandwidth samples would match the link

rate, and SQP would starve the elastic flow by utilizing the entire link bandwidth.

SQP paces each frame at a rate P , which is a multiple of the current bandwidth estimate, ie. $P = m \cdot B$ ($m > 1.0$). Thus, each frame is transmitted over $\frac{I}{m}$, where I is the frame interval. While pacing enables competing traffic to disperse SQP's packets, SQP's bandwidth samples would still be higher than the average rate it is sending at, and SQP would eventually starve the other flow. To avoid this problem, SQP combines frame pacing with a bandwidth target multiplier mechanism. SQP multiplies bandwidth samples with a target multiplier $T < 1$ before calculating the bandwidth estimate. SQP's target multiplier serves three key roles: (1) it allows SQP to drain any self-inflicted standing queues over time, (2) in conjunction with the pacing multiplier, it prevents SQP from starving competing flows, and (3) enables multiple SQP flows to converge to fairness. We empirically set $m = 2$ and $T = 0.9$, and analyze the impact of other values of T in Section § 5.1.

5 Analysis of SQP Dynamics

5.1 Competing Flows

SQP's pacing multiplier (m) and bandwidth target multiplier (T) mechanisms provide important guarantees that prevent SQP from starving other flows, and enable SQP to achieve fairness when competing with other SQP flows. In this section, we derive SQP's theoretical maximum share when competing with elastic flows, and the conditions under which SQP achieves queue-free operation when competing with inelastic flows. This analysis provides valuable insight into how SQP's parameters can be tuned for application-specific performance requirements.

SQP adds $B \cdot I$ bytes (i.e., the frame size, equal to the bandwidth estimate times the frame interval) to the bottleneck queue over a period $\frac{I}{m}$ (§ 4.5), during which a competing flow transmitting at a rate R adds $\frac{R \cdot I}{m}$ bytes to the queue. Thus, the time to drain the queue (T_d) is

$$T_d = \frac{B \cdot I + \frac{R \cdot I}{m}}{C} \quad (5)$$

where C is the link capacity. If the link is not being overutilized (Δ_{min} remains constant),

$$T_d = R_{end} - R_{start} = R_{end} - S_{start} - \Delta_{min} \quad (6)$$

From Eq. 6 and Eq. 1, SQP's bandwidth sample (S) can be written as $S = \frac{B \cdot I}{T_d}$. After substituting the value of T_d from Eq. 5 and simplifying the equation, we get:

$$S = \frac{C}{1 + \frac{R}{m \cdot B}} \quad (7)$$

Note that we assume $m \cdot B + R > C$ (link is not severely underutilized), otherwise no queue will build up during the SQP's pacing burst, and the bandwidth sample would simply be $m \cdot B$. SQP multiplies the bandwidth sample (S) with a target multiplier (T) before it is used to update the current bandwidth estimate using Eq. 4. Steady state occurs when the bandwidth estimate (B) is equal to the bandwidth target, ie. $B_T = S \cdot T$. Substituting S from Eq. (7),

we get:

$$B = C \cdot T - \frac{R}{m} \quad (8)$$

If $A = \frac{C-R}{C}$ is the fraction of the link capacity available for SQP, and $U = \frac{B}{C-R}$ is SQP's utilization of the available link capacity, Eq. 8 can be re-written as

$$U = \frac{m \cdot T + A - 1}{m \cdot A} \quad (9)$$

This equation predicts SQP's behavior in a variety of scenarios. Figure 5a plots U on the Y-axis as a function of A on the X-axis for various target multipliers and for a pacing multiplier of 2.

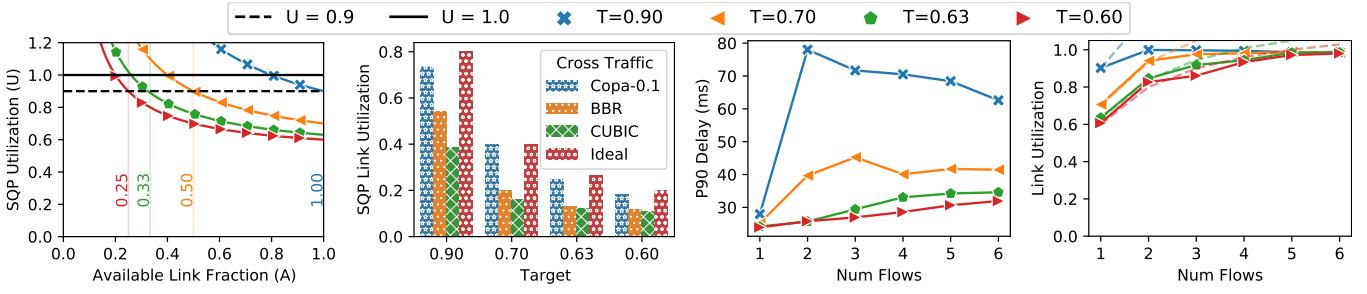
Recovery From Self-Induced Queuing When SQP is the only flow on a bottleneck link, the available link share $A = 1$ (right edge of Figure 5a). This implies that SQP will always underutilize the link slightly (specifically, it will use fraction T of the total link capacity), which will result in standing queues getting drained over time. The value of T caps SQP's maximum link utilization in the steady state, and determines how quickly SQP will recover from self-induced standing queues. In our evaluation and for SQP's deployment in Google's AR streaming service (§ 7), we use a target multiplier of 0.9, which achieves good link utilization and is able to drain standing queues reasonably well.

Inelastic Cross Traffic. When SQP competes with an inelastic flow (transmitting at a fixed rate), the available link fraction (A) is fixed. For SQP to operate without any queuing, U (SQP's utilization of the available capacity) must be less than 1. Thus, the available bandwidth must be greater than the value at which the utilization curve crosses $U = 1$ in Figure 5a.

For example, with a pacing multiplier of 2 and a target multiplier of 0.9, SQP requires at least 80% of the link to be available so that it can consistently maintain a slight underutilization of the link. When less than 80% of the link is available, SQP will tend to overutilize its share and cause queuing. While SQP's initial window size for tracking Δ_{min} (4.3) may not be large enough for SQP to completely drain the self-induced queue, the increase in the RTT due to queuing will eventually cause the window to grow to a size that is large enough to stabilize SQP's queuing. While a smaller target value would enable SQP to operate without queuing for lower values of A , it would sacrifice link utilization when there are no competing flows.

Elastic Cross Traffic. The minimum value of A for queue-free operation of SQP when competing with inelastic flows is also the maximum bound for SQP's share of the throughput when it is competing with elastic traffic. When SQP is not using its entire share ($U < 1$), the elastic flow will increase its own share since the link is underutilized. This reduces the available link share for SQP, moving the operating point to the left in Figure 5a until the entire link is utilized ($U = 1$). If SQP is over-utilizing its share, the elastic flow will decrease its own share and move the operating point to the right until the link is no longer being over-utilized.

This is an upper-bound of SQP's share. Non-ideal elastic flows can cause queuing delays that will cause SQP to increase its one-way delay tracking window, which in turn will make the bandwidth samples more sensitive to delay variations caused by the cross



(a) Theoretical utilization of available capacity.
(b) Single SQP flow throughput when competing with cross traffic.

(c) P90 packet delay for SQP flows sharing a bottleneck.
(d) Total link utilization for SQP flows sharing a bottleneck.

Figure 5: Impact of the target multiplier on delay, link utilization and link share obtained under cross traffic for pacing multiplier $m = 2$. Experimental results validate the theoretical analysis. In each case, the bottleneck link rate was 20 Mbps, the one-way delay in each direction was 20 ms and the bottleneck buffer size was 120 ms.

traffic. Figure 5b shows the share of a single SQP flow competing with various elastic flows, for different target multipliers. Copa-0.1 closely resembles an ideal elastic flow which does not cause queuing and has low delay variation. Hence, SQP's share (shown in blue) is close to the theoretical maximum (shown in red). With BBR and Cubic, SQP's share is less than the theoretical maximum since the higher delays induced by BBR and Cubic make SQP more reactive to queuing delay variations.

5.2 Intra-protocol Dynamics and Fairness

From the analysis in § 5.1, we can also infer the number of SQP flows that can operate without queuing on a shared bottleneck, with some caveats. The underlying assumption in § 5.1 is that packet arrivals at the bottleneck are evenly spaced. The analysis also holds in the case of Poisson arrivals since the bandwidth samples are smoothed out by the update rule (§ 4.4). Multiple SQP flows transmit frames as regularly spaced bursts, and thus, the packet dispersion observed by one SQP flow depends on how its frames align with the frames of the other flows. If the two flows that are sharing the bottleneck have perfectly aligned frame intervals, each flow will observe exactly half of the link rate, and they will operate without queuing since $T < 1$. If the frame intervals are offset exactly by $1/2$ (when pacing at 2X), each flow will see the full link bandwidth until link overutilization triggers SQP's transient delay recovery mechanism. When the intervals are offset by $1/4$, the packet dispersion is the same as the dispersion caused by a uniform flow. Note that this is only a concern if there are very few SQP flows, and the applications have perfectly timed frames. As the number of SQP flows increase, the aggregate traffic pattern gets smoothed out.

Figures 5c and 5d show the 90th percentile delay and the total link utilization respectively on the Y-axis as a function of the number of flows for various target multipliers. Figure 5d plots the theoretical link utilization of multiple SQP flows using dashed lines. The pacing multiplier was set to 2.0 for all runs. To avoid the impact of frame alignment in our experiment, we incorporate 1 ms of jitter into the frame generation timing⁵. When $T = 0.9$, a single SQP flow in isolation maintains low delay and utilizes 90% of the link; two

⁵Incorporating 1ms of sub-frame jitter into an application's frame rendering will have minimal impact on video smoothness

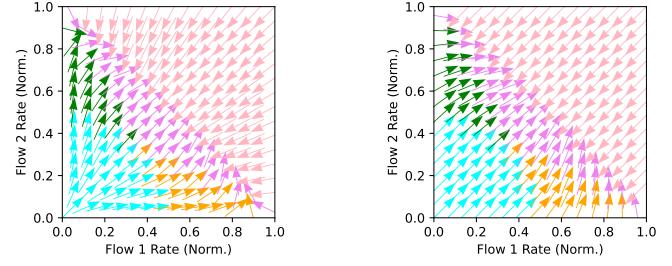


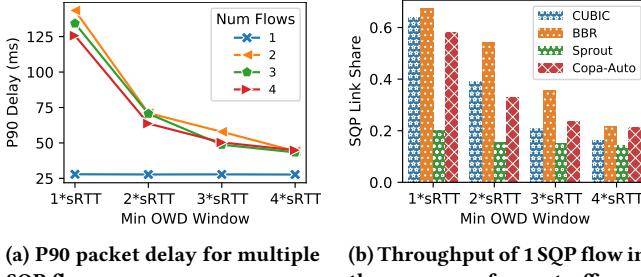
Figure 6: Vector field showing bandwidth update steps for different starting states for two competing flows. SQP's update rule significantly speeds up convergence to fairness.

or more SQP flows fully utilize the link and stabilize at a slightly higher delay (similar to SQP's behavior with inelastic cross-traffic, § 5.1). Reducing the target value reduces the steady state queuing delay, with the trade-off that an isolated SQP flow will have lower link utilization (Figure 5d) and will obtain less throughput share when competing with elastic flows (§ 5.1).

While SQP can be adapted to use more sophisticated mechanisms like dynamic frame timing alignment across SQP flows and dynamically lowering the target and pacing multipliers when the presence of multiple SQP flows is detected, we defer this to future work and only evaluate the base SQP algorithm with a fixed target multiplier $T = 0.9$ and a fixed pacing multiplier $m = 2$ in § 6.

Fairness. When competing with other SQP flows, there are two key mechanisms that enable SQP to converge to fairness: SQP's pacing-based bandwidth probing (§ 4.2), and SQP's logarithmic utility-based bandwidth smoothing (§ 4.4).

Let's consider a scenario where SQP is not using bandwidth smoothing, and directly updates its bandwidth estimate according to the sample. When overutilization occurs, each flow observes a common delay signal, and hence the bandwidth is reduced by a multiplicative factor. For various values of each flow's initial rate, we compute the update step as $S \times T - B$, where S is computed using Eq. 7 (average case behavior with randomized frame alignment, § 5.1). When the link is severely underutilized by a flow (the pacing burst of SQP does not cause queuing - see § 5.1), the update step is $2 \times B - B = B$. These update steps are shown in Figure 6a as arrows,



(a) P90 packet delay for multiple SQP flows. (b) Throughput of 1 SQP flow in the presence of cross traffic.

Figure 7: Impact of the min one-way delay multiplier on frame delay and throughput when competing with other flows. The bottleneck setup is the same as Figure 5, and $T = 0.9, m = 2$.

where the tail of the arrow is anchored at the initial condition, and the length of the arrow is proportional to the step size. In the cyan region, neither of the flows cause queuing due to their pacing burst, and hence, the rates undergo a multiplicative increase (direction of increase passes through the origin on the graph). In the purple region, both flows cause temporary queuing due to their pacing burst, and the slower flow increases its rate more than the faster flow (whose increase is sublinear). The green and orange regions depict a region of transition, where only one of the flows observes pacing-induced queuing. Thus, while SQP will undergo multiplicative increase when the link is severely underutilized, as the link utilization increases, the increases become sublinear. A downside is that convergence is slow when the link is severely underutilized - a faster flow will increase its rate more quickly as compared to a slower flow initially due to the multiplicative increase. We solve this using SQP's bandwidth update rule (§ 4.4), which significantly speeds up convergence to fairness.

In Figure 6b, we compute the update steps by incorporating SQP's logarithmic utility-based bandwidth update rule. In this case, SQP undergoes linear increase when the link is severely underutilized, sublinear increase when the link is close to being fully utilized, and linear decrease when the link is overutilized. Thus, SQP converges to fairness (similar to AIMD). The linear increase speeds up convergence for multiple SQP flows from a severely under-utilized state. The linear decrease makes SQP's throughput stable when competing with queue-building flows, since it does not react as fast as multiplicative decrease. SQP's bandwidth update rule also ensures that the updates are proportional to the difference relative to the current estimate, as opposed to fixed-size steps (e.g., additive increase in Cubic) or velocity-based mechanisms (e.g., Copa). We evaluate SQP's fairness in § 6.9.

5.3 Adaptive Min One-way Delay Tracking

SQP's adaptive min one-way delay window is a key mechanism that enables SQP to recover from network overutilization. Recall that SQP's window scales with the currently observed sRTT (§ 4.3). A larger window speeds up recovery from queuing caused by overutilization, but results in poor performance when competing with queue-building cross traffic. Different multipliers are evaluated in Figure 7. With $T = 0.9$ and $m = 2$, more than one SQP flows sharing a bottleneck require a larger Δ_{min} window to stabilize. A

multiplier of 2 results in acceptable level of steady state queuing (nearly as low as 3× and 4×), while achieving reasonable throughput in the presence of queue-building cross traffic like Cubic and BBR. SQP competing with Sprout is also shown as a worst case example; Sprout causes significant delay variation due to its bursty traffic pattern, causing SQP to achieve low throughput.

6 Evaluation

SQP's evaluation has three broad themes. § 6.4 evaluates SQP's performance on a large set of calibrated emulated links modeled after real-world network traces obtained from Google's game streaming service. §§ 6.5-6.10 evaluate SQP's throughput when competing with cross traffic, impact of shallow buffers, fairness, and bandwidth probing in application-limited scenarios. In § 7, we compare SQP and Copa (without mode switching) in the real world on Google's AR streaming service. In this section we compare SQP's performance to recently proposed high performance low latency algorithms like PCC [12], Copa [14] (with and without mode switching), Vivace [19] and Sprout [21], traditional queue-building algorithms like TCP-Cubic [11] and TCP-BBR [13], and WebRTC (using GoogCC as CCA), an end-to-end low-latency streaming solution.

6.1 Emulation Setup

We use the Pantheon [30] congestion control testbed, which works well for links under 100 Mbps. For the baselines, we use the implementations available on Pantheon. These include kernel-space (Linux) implementations of Cubic and BBR-v1 [38] (iperf3 [39]), user-space implementations of PCC [40], Vivace [41], Copa [42], and Sprout, and Chromium's version of WebRTC (with GoogCC, max bitrate changed to 50 Mbps from 2 Mbps). Additionally, we evaluate the Copa algorithm with a fixed delta ($\delta = 0.1$). We implement additional functionality in Pantheon, including flow-specific RTTs, start and stop times, and testing of heterogeneous CCAs sharing a link. For experiments with fixed bandwidth links, we choose a queue size of 10 packets / Mbps ($\approx 120\text{ms}$ for 20 Mbps) and the drop-tail queuing discipline. We fix $T = 0.9$ and $m = 2.0$ for SQP.

6.2 Metrics

While metrics like average throughput and packet delay are sufficient for evaluating a general purpose congestion control algorithm, they do not accurately reflect the impact on quality-of-experience (QoE) of a low-latency streaming application that is using a particular congestion control algorithm [43]. To evaluate how a CCA affects the QoE of low-latency streaming, we need metrics that quantify properties like video bitrate and frame delay.

After an experiment is run, Pantheon generates detailed packet traces with the timestamps of packets entering and leaving the bottleneck. We compute a windowed rate from the ingress packet traces, which serves as a baseline for the video bitrate. For a time slot t , the frame size $F(t)$ is:

$$F(t) = \max \left(\frac{S(t, t + n \cdot I) - p}{n}, S(t, t + I) - p, 0 \right) \quad (10)$$

where p denotes the pending unsent bytes from previous frames, I is the frame interval, $S(t_1, t_2)$ is the number of bytes sent by an algorithm between t_1 and t_2 and n is the window size in number

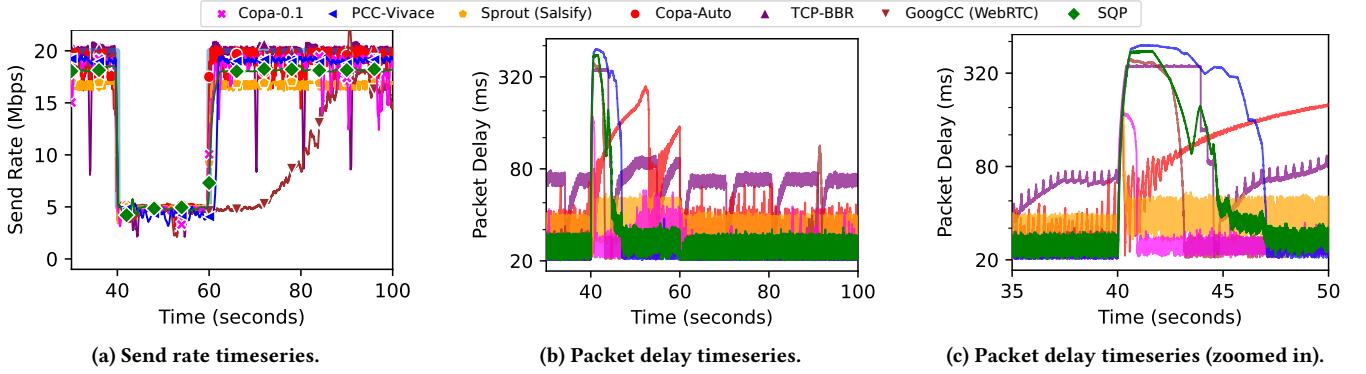


Figure 8: Congestion control performance on a variable link (link bandwidth shown as a shaded light blue line).

of frames used for smoothing. This ensures that none of the bytes the algorithm sent in a particular interval are wasted (maximum utilization).

To quantify video frame delay, we simulate the transmission of the frames to measure the end-to-end frame delay. For zero size frames, we assume that the delay of the frame is the time until the next frame. The choice of n limits the worst case sender-side queuing delay to n frames, which can occur when an algorithm sends a burst of packets during the n^{th} frame slot after a quiescence period of $n - 1$ frames.

6.3 Simple Variable Bandwidth Link

We evaluated SQP on a link that runs at 20 Mbps for 40 seconds, drops to 5 Mbps for 20 seconds, and then recovers back to 20 Mbps (same as the experiment described in § 3.1). The throughput is shown in Figure 8a, and the delay is shown in Figure 8b, with a zoomed version of the delay in Figure 8c. SQP quickly probes for bandwidth after the link rate increases ($T = 60$), and is able to maintain consistent, low delay when the link conditions are stable. When the link rate drops, SQP’s recovery is faster than PCC-Vivace, and as fast as BBR. While GoogCC’s recovery is slightly faster, it takes a very long time compared to SQP in order to ramp up once the link increases back to 20.

6.4 Real-world Wireless Traces

To evaluate SQP’s performance on links with variable bandwidth, delay jitter and packet aggregation, we obtained 100 LTE and 100 Wi-Fi throughput and delay traces from a cloud gaming service. Each network trace was converted to a MahiMahi trace using packet aggregation to emulate the delay variation, and the link delay was set to the minimum RTT for each trace.

Figures 9a and 9b show the throughput and delay of a single flow operating on a representative Wi-Fi trace. The thick gray line represents the link bandwidth. SQP achieves high link utilization and can effectively track the changes in the link bandwidth while maintaining low delay. While Copa-Auto, Sprout, and BBR achieve high link utilization, they incur a high delay penalty. WebRTC, PCC and Vivace are unable to adapt to rapid changes in the link bandwidth, resulting in severe link underutilization and occasional delay spikes (e.g., Vivace at $T=22s$).

The aggregated results for the Wi-Fi traces are shown in Figure 10a. Across all Wi-Fi traces, SQP achieves 78% average link

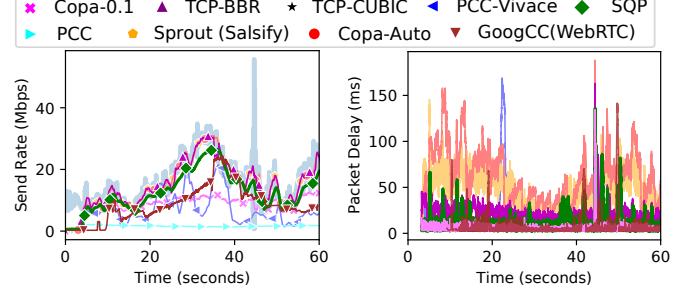


Figure 9: Performance of various CCAs on a sample Wi-Fi network trace, with the bottleneck buffer size set to 200 packets. SQP rapidly adapts to the variations in the link bandwidth, and achieves low packet queueing delay.

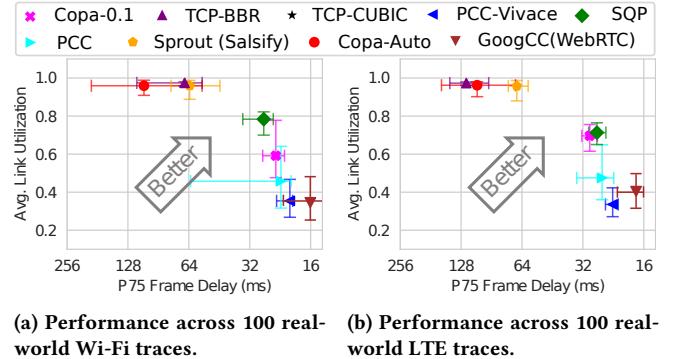


Figure 10: SQP’s performance over emulated real-world wireless network traces. The bottleneck buffer size was set to 200 packets. In Figures 10a and 10b, the markers depict the median across traces and the whiskers depict the 25th and 75th percentiles.

utilization compared to 46%, 35% and 59% for PCC, Vivace and Copa-0.1 respectively while only incurring 4-8 ms higher delay. While Cubic, BBR, Sprout, and Copa-Auto achieve higher link utilization, this is at a cost of significantly higher delay (130-342% higher).

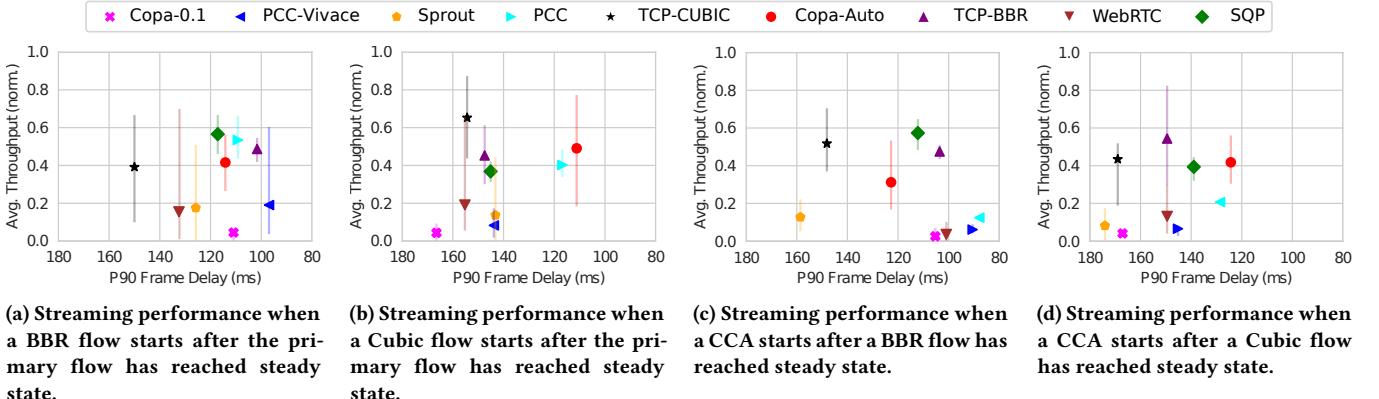


Figure 11: CCA performance when competing with queue-building cross traffic. The error bars mark the P10 and P90 simulated frame bitrates (§ 6.2).

Figure 10b shows the performance of various CCAs across 100 real-world LTE traces. SQP and Copa-0.1 have good throughput and delay characteristics, whereas other CCAs either have very high delays or insufficient throughput.

6.5 Competing with Queue-building Flows

Next, we evaluate the ability of various congestion control algorithms to support stable video bitrates in the presence of queue-building cross traffic. We ran the experiment for 60 seconds on a 20 Mbps bottleneck link with 120 ms of packet buffer, and a baseline RTT of 40 ms, where each algorithm is run for 10 seconds before the cross traffic is introduced. Figure 11a shows the average normalized throughput and P10-P90 spread of the windowed bitrate for each algorithm versus the P90 simulated frame delay after a BBR flow is introduced. Figure 11c shows the average normalized throughput versus the P90 simulated frame delay when the CCA being tested starts 10 seconds after a BBR flow is already running on the link. We ran similar experiments with Cubic as the cross traffic, and the results are shown in Figures 11b and 11d.

SQP is able to achieve high and stable throughput due to SQP's bandwidth sampling mechanism (§ 4.2) and the use of a dynamic min-oneway delay window size (§ 4.3). While PCC performs well when it starts before the competing traffic is introduced on the link, PCC's normalized throughput is less than 0.2 when it starts on a link that already has a BBR or Cubic flow running on it. GoogCC's slower start affects its throughput, with things improving slightly if the Cubic flow starts after 20s (App. B), and it is also suffers from the latecomer effect. Vivace, Copa-0.1 and Sprout are unable to maintain high throughput in all the cases. While Copa-Auto has good average throughput, its performance is unstable at the frame timescale, which is evident by the spread between the P10 and P90 bitrate.

6.6 Shallow Buffers

For the target workload of interactive video with $I = 16.66$ ms (60FPS) and a pacing multiplier $m = 2$, SQP's pacing-based bandwidth probing only requires approximately 8 ms of packet buffer at the bottleneck link to be able to handle the burst for each frame. The lines in Figure 12a show the link egress rate for various CCAs

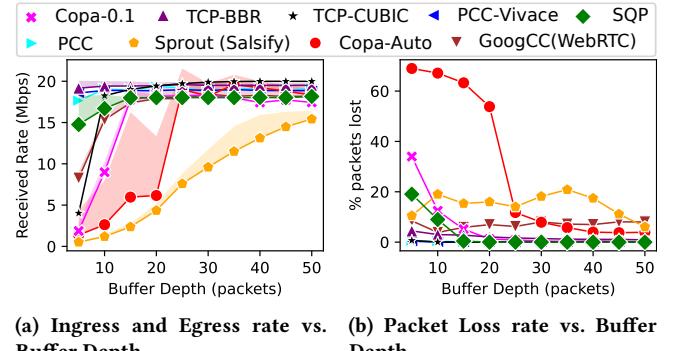


Figure 12: Performance impact of shallow buffers on a 20 Mbps, 40ms RTT link.

for different buffer sizes, and the shaded regions denote the rate of loss (i.e., the top of the shaded region is the link ingress rate). The loss rate is also shown in Figure 12b. SQP achieves its maximum throughput with a buffer of 15 packets or more, which corresponds to 8 ms of queuing on a 20 Mbps link. If the buffer size is smaller than 15 packets, SQP transmits at 18 Mbps ($T = 0.9$ fraction of the link capacity), but the packets that correspond to the tail end of each frame are lost. Copa-Auto, Sprout and GoogCC require larger buffers, whereas BBR (~4% loss with a 5 packet buffer), and both PCC versions (<1% loss with a 5 packet buffer) excel at handling shallow buffers. Typical last-mile network links like DOCSIS, cellular, and Wi-Fi links have much larger packet buffers [44]. When SQP competes with other flows (vs. SQP, inelastic flows), SQP may require a higher level of queuing to stabilize. Dynamic pacing and target mechanisms are required to handle such scenarios, and we leave that for future work.

Discussion: While SQP causes sub-frame queuing since it paces each frame at 2X of the bandwidth estimate, this queuing is limited to a maximum of 8 ms (14 packets for 20 Mbps). Hence, for buffer sizes of 15 packets and above, SQP has exactly zero loss. Sprout on the other hand has 10-20 % loss for buffer sizes all the way up to 50 packets, and more than 60% of packets sent by Copa-Auto are lost for buffer sizes lower than 20 packets. GoogCC (WebRTC)

has around 10% packet loss for the entire range of buffer sizes evaluated here, which may be due to WebRTC sending FEC packets in response to the loss observed.

6.7 Short Timescale Variations

In Figure 13, we show the short-term throughput and delay behavior of SQP and other various CCAs, over a period of 0.5 seconds (see § 3.2). Figure 13a shows the transmission rate for each frame period (16.66 ms at 60 FPS). SQP’s transmission rate is very stable, and does not vary at all across multiple frames. Figure 13b shows the packet delay for various CCAs over 0.5 seconds. Since SQP transmits each packet as a short (paced) burst, it causes queuing at sub-frame timescales, but since SQP does not use more than 90% of the link (due to $T = 0.9$, § 4.5), there is no queue buildup that occurs across frames. While sprout’s probing looks similar, the queuing caused by Sprout is much higher. We note that Sprout’s dips in throughput may be caused by the fact that the burst frequency of the Sprout sender used in our test is 50 cycles per second. This may not be a factor for video streaming if the burst frequency matches the video frame rate. Sprout’s inadequacy for low-latency interactive streaming applications primarily stems from its inability to achieve sufficient bandwidth when competing with other queue-building flows.

6.8 Impact of Feedback Delay

Since SQP receives the frame delivery statistics at the sender after 1-RTT, it is important to evaluate the impact of delayed feedback on SQP’s dynamics. Figure 14a shows the average delay, and Figure 14b shows the average throughput after a 20 Mbps link steps down to 5 Mbps, for various baseline network RTT values. The link is run at 20 Mbps for 40 seconds, following which the link is run at 5 Mbps for an additional 20 seconds. Figure 14a shows the average delay for the last 20 seconds, when the link is operating at 5 Mbps. SQP’s performance is consistent across the entire range, even though the feedback is delayed, and can be attributed to the fact that SQP uses a larger window for Δ_{min} on higher RTT links. Sprout and Copa-Auto have lower delay on higher RTT links, but for different reasons: Sprout’s link utilization drops sharply as the link RTT increases from 40 to 80 ms (Figure 14b), and hence, its delay is lower, whereas Copa-Auto incorrectly switches to competitive mode on low RTT links, causing very high delays (Figure 14c shows the delay timeseries for a 10ms RTT link). PCC-Vivace can only maintain low delay across a 10ms RTT link, and PCC is unable to drain the queuing caused after the link rate drops in all cases.

6.9 Fairness

The first experiment evaluates the performance of 10 homogeneous flows sharing a 60 Mbps bottleneck link, with a link RTT of 40 ms. Figure 15a compares the average throughput and the P90 one-way packet delay for each flow. The ideal behavior is that each flow achieves exactly 6 Mbps and low delay, ie. the points should be clustered at the 6 Mbps line and be towards the right in the plot. SQP flows⁶ achieve equal share of the link, with lower P90 one-way packet delay compared to Sprout, PCC, BBR and Cubic (75 ms). While BBR, and both versions of Copa achieve good fairness with

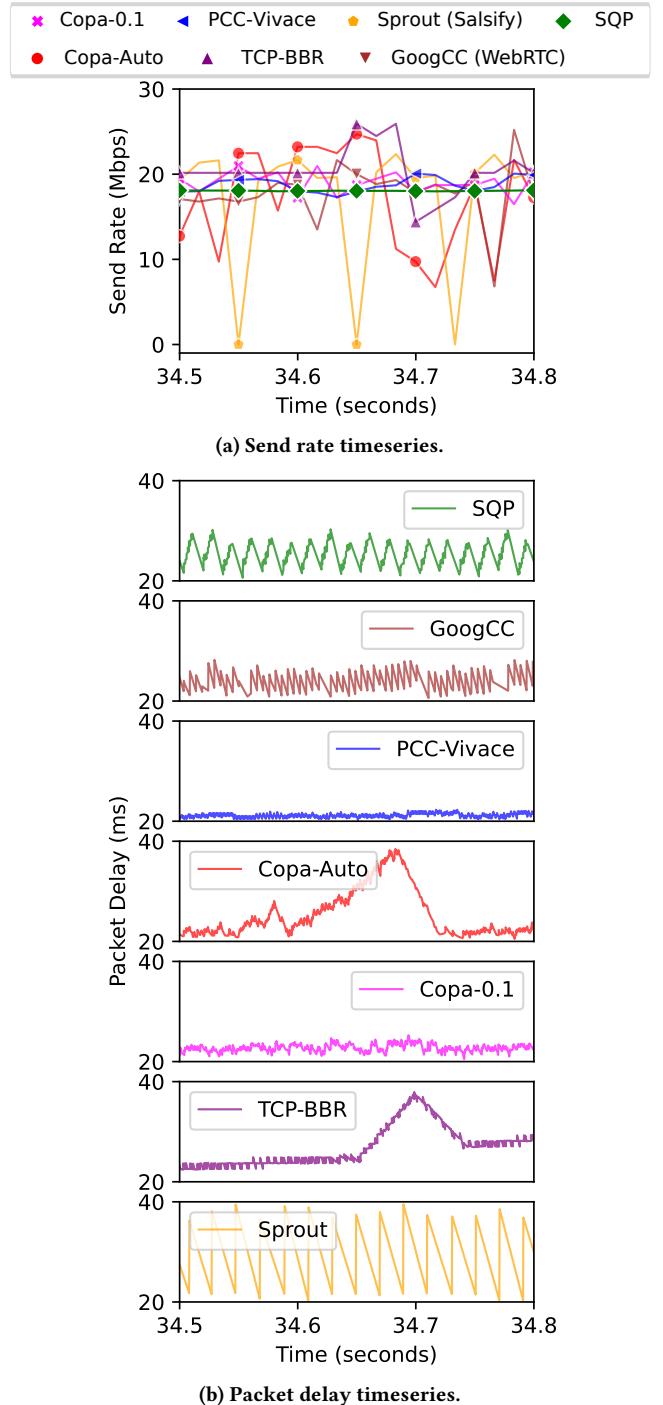


Figure 13: Short-timescale throughput and delay behavior on a 20 Mbps link (link bandwidth shown as a shaded light blue line).

⁶Inter-frame timing jitter enabled (§ 5.1)

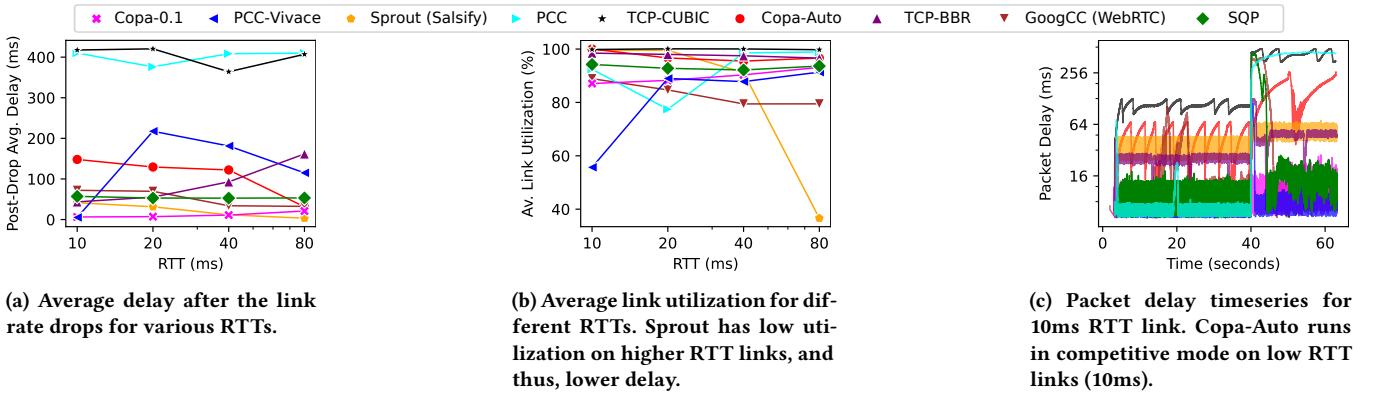


Figure 14: Impact of link RTT on throughput and delay, where link changes from 20 Mbps to 5 Mbps at T=40s.

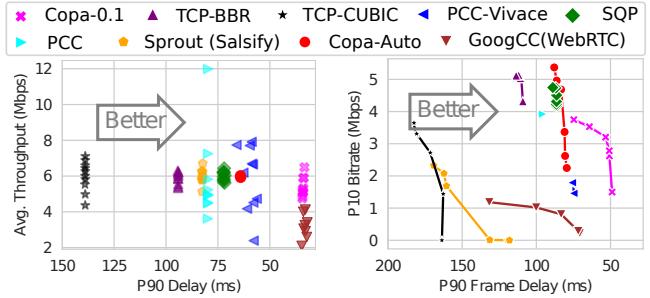


Figure 15: Fairness results with 10 flows sharing a bottleneck. SQP achieves a fair share of throughput on average, and at smaller time-scales. CCAs like Sprout, WebRTC, and Copa become excessively bursty at smaller time-scales.

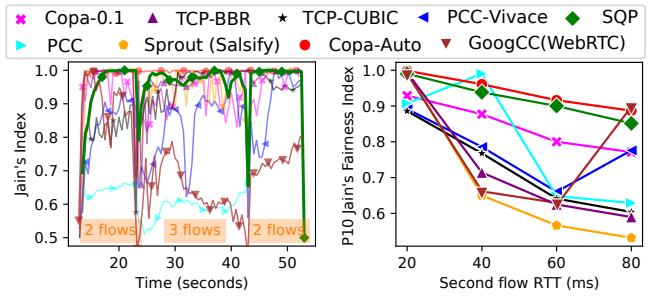


Figure 16: Dynamic fairness and RTT fairness comparison. SQP quickly converges to fairness, and has good RTT fairness.

respect to the average throughput for the full experiment duration, neither version of PCC is able to do so. While WebRTC has very low P90 packet delay, the flows cumulatively underutilize the link and do not achieve fairness. Figure 15b compares the streaming performance of the algorithms by plotting the P10 bitrate and the

P90 frame delay for different bitrate estimation windows ranging from 1 frame to 32 frames in multiplicative steps of 2 (§ 6.2). The streaming performance of Copa-0.1, Sprout, Cubic and WebRTC are significantly worse than their average throughput and packet delay due to bursty transmissions when multiple flows share the bottleneck link.

In the second experiment, we evaluate dynamic fairness as flows join and leave the network. Flows 2 and 3 start 10 s and 20 s after the first flow respectively, and stop at 40 s and 50 s respectively. Figure 16a plots the Jain fairness index [45] computed over 500 ms windows versus time. SQP converges rapidly to the fair share, whereas both versions of PCC, Copa-0.1 and WebRTC cannot reliably achieve fairness at these time scales.

SQP also demonstrates good fairness across flows with different RTTs. We evaluated steady-state fairness among flows that share the same bottleneck, but have different network RTTs. In Figure 16b, we plot the P10 fairness (using Jain's fairness index) across windowed 500 ms intervals. When two flows have the same RTT, SQP, Copa-Auto, TCP-BBR and Sprout achieve perfect fairness. As the RTT of the second flow increases, SQP and Copa-Auto are able to maintain reasonable throughput fairness but the fairness degrades rapidly in the case of TCP-BBR, Cubic and Sprout as the RTT of one flow increases. The slight drop in fairness in the case of SQP is because the flow with the higher RTT achieves lower throughput since its minimum one way delay window size is larger. PCC, Vivace, and WebRTC also achieve low fairness for flows with different RTTs and do not demonstrate any particular pattern as the RTT difference between the flows increases.

6.10 SQP Video Codec Integration

We evaluated SQP's bandwidth estimation in a scenario where the video bitrate is significantly lower than the bandwidth estimate. We tested SQP by artificially limiting the video bitrate on a 20 Mbps, 40 ms RTT link with 120 ms of bottleneck buffer. The encoder bitrate is artificially capped to 2 Mbps for three 2-second intervals. In Figure 17a, SQP maintains a high bandwidth estimate, which is appropriate since SQP is the only flow on the link. SQP also obtains a reasonable estimate of the link bandwidth under application-limited scenarios when competing with other flows. Figure 17b shows SQP's bandwidth estimate when the video bitrate is lower than the target bitrate and SQP is competing with a Cubic flow. When the

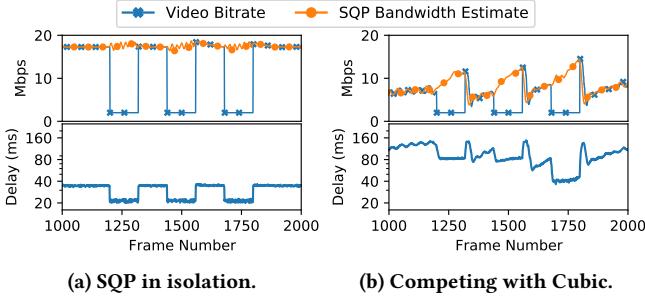


Figure 17: SQP’s performance when application-limited.

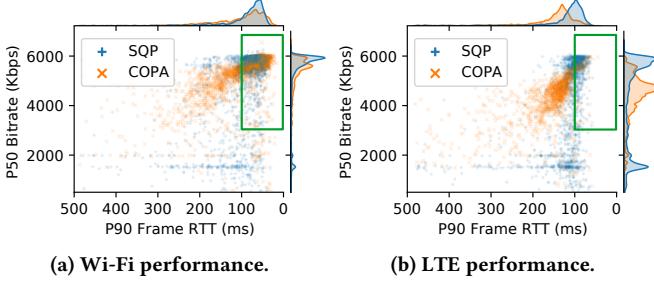


Figure 18: Real world A/B testing of SQP and Copa-0.1.

video bitrate is lower than the target, SQP is able to maintain a high bandwidth estimate, which demonstrates that SQP is able to maintain a high bandwidth estimate without requiring additional padding data. This allows SQP to quickly start utilizing its share when the video bitrate is no longer limited (matches the target bitrate), instead of acquiring its throughput share from scratch, which would take much longer. These experiments demonstrate that padding bits are not necessary for SQP to achieve good link utilization.

The generated video bitrate can also overshoot the requested target bitrate. In such scenarios, it is typically the encoder’s responsibility to make sure that the average video bitrate matches the requested target bitrate, although SQP can handle and recover from occasional frames size overshoots since they would cause subsequent bandwidth samples to be lower. Persistent overshoot can occur in very complex scenes when the target bitrate is low. In such cases, the application must take corrective actions that include reducing the frame rate or changing the video resolution. Salsify [29] proposes encoding each frame at two distinct bitrates, choosing the most appropriate size just before transmission. While SQP can serve as a viable replacement for Sprout in Salsify, in Appendix A, we show that modern encoders like NVENC [46] have good rate control mechanisms that avoid overshoot and can consistently match the requested target bitrate.

7 Real-World Performance

To evaluate SQP’s performance in the real world, we deployed SQP in Google’s AR streaming platform. We also deployed Copa-0.1

(without mode switching) on the same platform by adapting the MVFST implementation of Copa [20] and performed A/B testing, comparing the performance of the two algorithms. We chose Copa-0.1 since it consistently maintained low delay compared to other CCAs (e.g. Sprout (Salsify) has very high delays) on emulated tests, and has been demonstrated to work well for low-latency live video in a production environment [32]⁷. For Copa, we use $\frac{CWND}{sRTT}$ to set the encoder bitrate, and reduce the bitrate by $\frac{Q_{sender}}{D}$ when sender-side queuing occurs (Q_{sender} = pending bytes from previous frames, $D = 200$ ms is a smoothing factor), gradually reducing the sender-side queue over a period of 200 ms. We ran the experiment for 2 weeks and obtained data for approximately 2400 Wi-Fi sessions and 1600 LTE sessions for each algorithm. Figure 18 shows the scatter plots of the median bitrate and the P90 frame RTT (fRTT; send start to notification of delivery) in addition to the separate distributions for each metric. 64 SQP and 105 Copa sessions over LTE, and 36 SQP and 52 Copa sessions over Wi-Fi had a P90 fRTT higher than 500 ms, and these are not shown in the figure.

71% of SQP sessions over Wi-Fi had good performance (bitrate > 3 Mbps, fRTT < 100 ms), compared to 56% of Copa-0.1 sessions. On LTE links, 36% of SQP sessions had good performance, compared to 9% of Copa sessions. Across all the sessions, fRTT was less than 100 ms for 64% of SQP sessions and only 39% of Copa sessions. These regions are highlighted with green boxes in Figure 18.

SQP achieves lower frame delay compared to Copa across both Wi-Fi and LTE. SQP on Wi-Fi also achieves higher bitrate compared to Copa. On LTE connections, SQP demonstrates a bi-modal distribution of the bitrate, with a significant number of sessions being stuck at a low bitrate despite having a low RTT. We believe SQP gets stuck at a low bandwidth estimate due to a combination of noisy links, a low bandwidth estimate and encoder undershoot, although this needs to be investigated further (Eq. 2 was not used). On the other hand, the bitrates for Copa sessions over LTE are more evenly distributed, but also incur higher delays compared to SQP.

Our results from emulation and real-world experiments demonstrate that SQP can efficiently utilize wireless links with time-varying bandwidth and simultaneously maintain low end-to-end frame delay, making it suitable for wireless AR streaming and cloud gaming applications.

8 Conclusion

In this paper, we have presented the design, evaluation, and results from real-world deployment of SQP, a congestion control algorithm designed for low-latency interactive streaming applications. SQP is designed specifically for low-latency interactive video streaming, and makes key application-specific trade-offs in order to achieve its performance goals. SQP’s novel approach for congestion control enables it to maintain low queuing delay and high utilization on dynamic links, and also achieve high throughput in the presence of queue-building cross traffic like Cubic and BBR, without the caveats of explicit mode-switching techniques.

⁷In addition, Salsify’s custom software encoder cannot sustain the frame rates required for low-latency interactive streaming applications

References

- [1] M. Abdallah, C. Griwodz, K.-T. Chen, G. Simon, P.-C. Wang, and C.-H. Hsu, "Delay-sensitive video computing in the cloud: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 14, no. 3s, Jun. 2018. [Online]. Available: <https://doi.org/10.1145/3212804>
- [2] "Amazon luna: Amazon's cloud gaming service." [Online]. Available: <https://www.amazon.com/luna/landing-page>
- [3] "Stadia." [Online]. Available: <https://stadia.google.com/>
- [4] "Cloud gaming (beta) with xbox game pass: Xbox." [Online]. Available: <https://www.xbox.com/en-US/xbox-game-pass/cloud-gaming/home>
- [5] "Geforce now gaming anywhere & anytime." [Online]. Available: <https://www.nvidia.com/en-us/geforce-now/>
- [6] "Deploy and scale your virtualized windows desktops and apps on azure." [Online]. Available: <https://azure.microsoft.com/en-us/free/services/virtual-desktop>
- [7] "Chrome remote desktop." [Online]. Available: <https://remotedesktop.google.com/>
- [8] "Streaming augmented reality with google cloud." [Online]. Available: <https://cloud.google.com/blog/products/networking/google-cloud-streams-augmented-reality>
- [9] "Azure mixed reality cloud services overview - mixed reality." [Online]. Available: <https://docs.microsoft.com/en-us/windows/mixed-reality/develop/mixed-reality-cloud-services>
- [10] "Nvidia cloud xr." [Online]. Available: <https://www.nvidia.com/en-us/design-visualization/solutions/cloud-xr/>
- [11] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
- [12] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 395–408.
- [13] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.
- [14] V. Arun and H. Balakrishnan, "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.
- [15] P. Goyal, A. Narayan, F. Cangialosi, D. Raghavan, S. Narayana, M. Alizadeh, and H. Balakrishnan, "Elasticity detection: A building block for delay-sensitive congestion control" in *ANRW*, 2018, p. 75.
- [16] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication," Internet Engineering Task Force, Internet-Draft *draft-ietf-rmcat-gcc-02*, Jul. 2016, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02>
- [17] G. Carlucci, L. De Cicco, S. Holmer, and S. 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*, 2016, pp. 1–12.
- [18] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman, "Performance evaluation of webrtc-based video conferencing," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, p. 56–68, mar 2018. [Online]. Available: <https://doi.org/10.1145/3199524.3199534>
- [19] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Pcc vivace: Online-learning congestion control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 343–356.
- [20] Facebookincubator, "facebookincubator/mvfst." [Online]. Available: <https://github.com/facebookincubator/mvfst/blob/master/quic/congestion-control/Copah>
- [21] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 459–471.
- [22] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.
- [23] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "{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.
- [24] Y. Zhang and T. R. Henderson, "An implementation and experimental study of the explicit control protocol (xcp)," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2. IEEE, 2005, pp. 1037–1048.
- [25] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan et al., "Swift: Delay is simple and effective for congestion control in the datacenter," 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. 514–528.
- [26] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.
- [27] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "Tcp lola: Congestion control for low latencies and high throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. IEEE, 2017, pp. 215–218.
- [28] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications*, 1994, pp. 24–35.
- [29] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. 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)*, 2018, pp. 267–282.
- [30] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 731–743.
- [31] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for http," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 417–429.
- [32] N. Garg, "Copa congestion control for video performance," Mar 2020. [Online]. Available: <https://engineering.fb.com/2019/11/17/video-engineering/copa/>
- [33] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 43–56. [Online]. Available: <https://doi.org/10.1145/347059.347397>
- [34] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar et al., "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.
- [35] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology," *IEEE/ACM Transactions On Networking*, vol. 12, no. 6, pp. 963–977, 2004.
- [36] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing tcp's retransmission timer," rfc 2988, November, Tech. Rep., 2000.
- [37] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, vol. 49, no. 3, pp. 237–252, 1998.
- [38] "Bbrv1: Linux kernel." [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next/git/tree/net/ipv4/tcp_bbr.c
- [39] A. Tirumala, "Iperf: The tcp/udp bandwidth measurement tool," <http://dast.nlanr.net/Projects/Iperf/>, 1999.
- [40] "PCC," <https://github.com/modong/pcc>, 2016.
- [41] "Vivace," <https://github.com/PCCproject/PCC-Uspace/tree/NSDI-2018>, 2016.
- [42] "genericCC," <https://github.com/venkatrun95/genericCC>, 2018.
- [43] Y. Liu and J. Y. Lee, "Streaming variable bitrate video over mobile networks with predictable performance," in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, 2016, pp. 1–7.
- [44] J. Gettys, "Bufferbloat: Dark buffers in the internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, 2011.
- [45] R. K. Jain, D.-M. W. Chiu, W. R. Hawe et al., "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.
- [46] A. Patai and E. Young, "High performance video encoding with nvidia gpus," in *2016 GPU Technology Conference (<https://goo.gl/Bdjgdm>)*, 2016.
- [47] [Online]. Available: https://www.lcevc.org/wp-content/uploads/Evaluation_of_MPEG-5_Part-2_LCEVC_for_Gaming_Video_Streaming_Applications_VQEG_CGI_2021_131.pdf
- [48] "Nvenc reference manual." [Online]. Available: <https://docs.nvidia.com/video-technologies/video-codec-sdk/nvenc-video-encoder-api-programming-guide/#recommended-nvenc-settings>

Appendix A Video Encoder

Overshoot/Undershoot

Previous systems like Salsify have focused on the challenges of integrating a video codec with the congestion control algorithm due to mismatches that can occur between encoder output and the requested rate. Salsify proposes a custom video encoder that encodes video frames at two different target bitrates, and chooses the largest frame whose transmission will not exceed the available network bandwidth. Since SQP does not have an explicit mechanism to handle overshoots (the bandwidth estimate is penalized in SQP due to transient queuing, but this may not be enough), SQP must rely on the accuracy of the video encoder’s rate control to produce accurate frame sizes. We evaluate the rate-control accuracy (whether the encoder is able to produce frames close to the requested target size) of a modern commercial off-the-shelf hardware video encoder, NVENC [46], that natively supports real-time interactive video streaming workloads. We use the videos from a cloud gaming video data set (CGVDS [47], 1080p@60FPS, 30s duration) for evaluating the encoder’s rate-control accuracy. We encode the video by randomly changing the target bitrate for each frame, where the bitrate is sampled from a uniform distribution between [2 Mbps, 20 Mbps]. The NVENC settings were chosen according to the values specified in the official NVENC documentation [48] for low-latency video streaming.

The results are shown in Figure 19b. The X-axis is the requested target bitrate, and the Y-axis plots the 90th percentile percentage frame size delta ($100 \cdot \frac{\text{actual}-\text{requested}}{\text{requested}}$) for that bitrate for various videos. For bitrates including 5 Mbps and above, the encoder does an excellent job of keeping the video bitrate under the target bitrate.

Some overshoot occurs at lower rates (≤ 4 Mbps). In Figure 19c, we plot the CDF of the frame size delta for target bitrates 4 Mbps and under. In this case, we observe that the encoder is still able to do a good job, and only occasionally overshoots the target bitrate. Worms_30s and LoL_TF_30s have the highest fraction of frames that overshoot the target bitrate, but this fraction is also low (around 15%).

Our conclusion is that with modern encoders like NVENC, in practice, video bitrate overshoots only happen at the lowest bitrates, and are otherwise not a major concern. Overshoot can be attributed to there just being too much data to encode in the video, and not rate control accuracy specifically, and solutions may include reducing the video resolution in order to accommodate the lower bitrates. While techniques like Salsify [29] are useful for low bitrate operation, where accurately controlling frame sizes is critical, and challenging, modern hardware-based codecs already do a good job at controlling video bitrate overshoot at bitrates commonly used for applications like cloud gaming and AR streaming.

Video bitrate undershoots are more common, and SQP is able to handle these scenarios well (§ 6.10, , § 6.10).

Appendix B GoogCC startup behavior

In § 6.5, we start the competing Cubic flow after 10 seconds, but GoogCC does not converge to its maximum throughput at steady state by that point. Thus, the throughput is slightly lower than if it had reached steady state. Figure 20 shows the throughput and delay for the same experiment, but we vary the start time of the competing flow between 5 and 20 seconds. While GoogCC’s throughput is high for a short period of time right after the Cubic flow starts, the GoogCC flows eventually converge to the same rate.

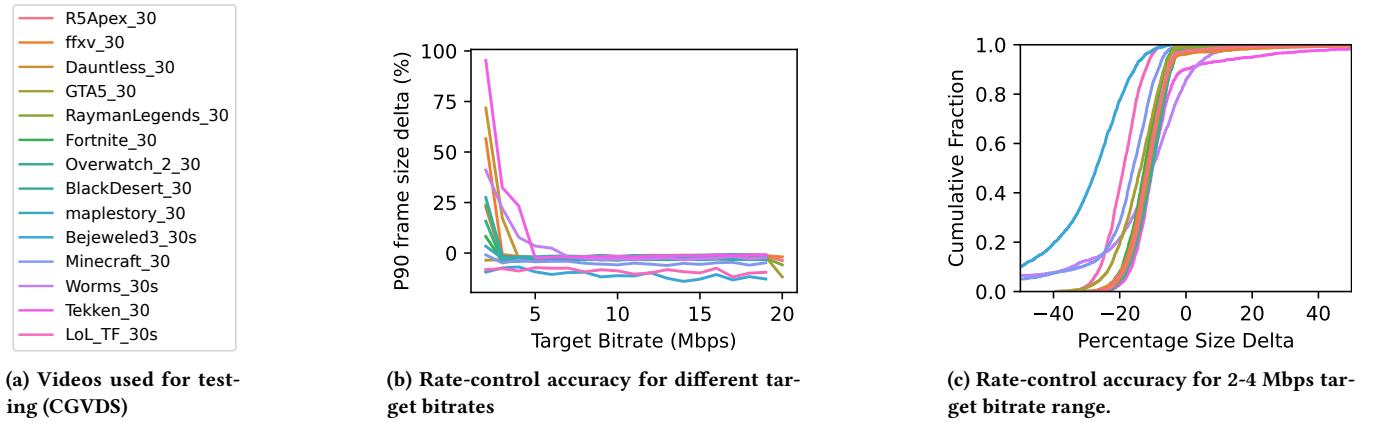


Figure 19: Rate-control accuracy of the NVENC encoder, tested in low-latency configuration.

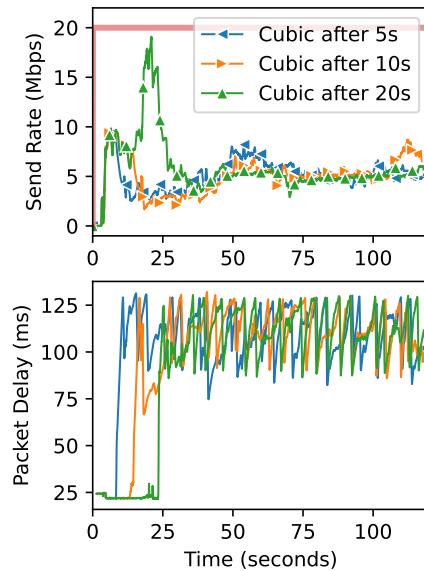


Figure 20: GoogCC's slower startup affects short-term throughput when competing with Cubic.