Mortise: Auto-tuning Congestion Control to Optimize QoE via Network-Aware Parameter Optimization

Yixin Shen[1,2,3], Ruihua Chen[1], Bo Wang[1,3], Jing Chen[1], Haochen Zhang[1], Minhu Wang[1], Yan Liu[2], Mingwei Xu[1,3], Zili Meng[4]
[1]*Tsinghua University,* [2]*Bytedance Inc.,* [3]*Zhongguancun Laboratory,* [4]*HKUST*

Abstract

Congestion control algorithms (CCAs) critically shape the tradeoff among throughput, latency, and loss, directly impacting user Quality of Experience (QoE). However, most existing CCAs use *static*, *heuristically* chosen parameter settings that fail to adapt to dynamic network states, resulting in suboptimal QoE. Our key observation is that the optimal CCA parameter configuration depends on real-time network states. To bridge this gap, we propose Mortise, a real-time, network-aware adaptation framework that dynamically tunes rule-based CCA parameters to maximize QoE. To address the challenges in modeling the complex parameter-QoE relationship, Mortise introduces a QoS tradeoff proxy to decompose parameter optimization into two steps: it first infers the application's preferred QoS tradeoff from real-time QoE gradients and then derives the corresponding parameter settings via control-theoretic analysis. Implemented atop TCP and evaluated in both emulated and production environments, Mortise outperforms state-of-the-art solutions, enhancing the QoE of file downloading service by up to 73% and QoE of video streaming service by up to 167% in real-world scenarios, with minimal deployment overhead.

1 Introduction

The Quality of Experience (QoE) perceived by end users is critical as it directly impacts user retention and the economic outcomes of content providers. While traditional QoE optimization primarily relies on application-layer mechanisms such as adaptive bitrate algorithms [5, 34] and content delivery network selection [40, 49, 61], recent studies [32, 45, 60] highlight the pivotal role of transport-layer congestion control algorithms (CCAs) in shaping QoE. CCAs govern the tradeoffs among key network Quality of Service (QoS) metrics — throughput, latency, and packet loss — which fundamentally determine user-perceived performance, including video stalling and interactive delay. Consequently, modern CCA research has increasingly focused on QoE-driven design.

Whether rule-based [12, 22] or optimization-based [15, 16], the performance of CCAs hinges on critical parameters. In rule-based schemes, parameters such as target queue length and window increase/decrease granularity determine how aggressively the sender probes for bandwidth or backs off under congestion. In optimization-based CCAs, the relative weights assigned to throughput, delay, and loss in the objective function steer the algorithm's operating point on the Pareto frontier of achievable performance [50, 59]. Because the network state fluctuates continuously, any CCA has to navigate in-

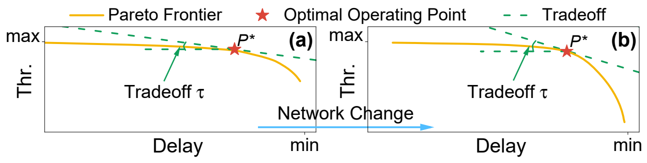


Figure 1: An illustration of the optimal operating point shifting on the Pareto frontier with network state.

trinsic trade-offs among the different QoS metrics. Existing solutions heuristically preconfigure the algorithm parameters offline, producing static QoS tradeoffs tailored to specific applications. For instance, latency-sensitive interactive applications choose small queue targets or high delay-penalty weights, whereas bulk transfers prioritize high throughput through throughput-emphasis objectives.

Despite these efforts, existing *heuristic* and *static* configurations fail to maximize QoE in dynamic networks due to two key limitations: (1) *Heuristic parameter selection*: Most CCAs rely on empirically tuned parameters without rigorously modeling the complex relationship between these parameters and QoE. This often results in suboptimal tradeoffs in QoS metrics that misalign with actual QoE requirements. (2) *Static configuration*: Using fixed parameters throughout an entire session ignores the fact that the network states vary largely over time, which can have an impact on the parameter settings. Without dynamic parameter adaptation, it can prevent a consistently high QoE.

Our key observation is that *the optimal configuration of CCA parameter for maximizing QoE inherently depends on real-time network states*. This is because the application's preferred balance among QoS metrics shifts as the network states evolve. For example, in interactive video streaming, when bandwidth is abundant, further increasing throughput yields diminishing QoE gains. Instead, prioritizing a shorter network queue to reduce playback stall risk can yield a greater improvement in user satisfaction. Thus, the optimal point on the QoS Pareto frontier is not static but instead dynamically moves with the network state. *To consistently provide optimal QoE, frequent and precise parameter adjustments are required throughout a session.*

However, frequent and optimal tuning of CCA parameters faces two key challenges:

- **Challenge in identifying optimal parameters:** The relationship among CCA parameters, network states, and QoE performance is highly complex and context-dependent. For example, determining how to set Copa's target queue length to minimize video stalling time, or identifying PCC's optimal weights for throughput, latency, and loss to reduce flow completion time, requires solving high-dimensional, non-convex optimization problems. Exhaustive search is computationally prohibitive, and existing approaches lack quantitative, interpretable models to predict how parameter adjustments affect QoE.

- **Challenge in real-time behavior adaptation:** Rule-based CCAs lack explicit mechanisms to translate desired QoS tradeoffs into corresponding parameter settings. While learning-based approaches facilitate QoS tradeoff adjustments by modifying optimization objectives, they often suffer from significant training and inference delays, limiting their ability to update policies in response to real-time parameter changes promptly.

To address these challenges, we propose Mortise, a CCA parameter adaptation framework that continuously optimizes CCA configurations to maximize QoE in time-varying network conditions. Mortise builds on rule-based CCAs to support flexible and real-time adaptation, dynamically tuning parameters (e.g., target queue length, sending rate probe speed) to maintain QoE-optimal operating points as network states evolve. Rather than directly modeling the complex parameter-QoE relationship, Mortise introduces a QoS tradeoff proxy that decomposes the problem of identifying the optimal configuration into two tractable sub-tasks. The underlying rationale is that operating at the optimal QoS tradeoff enables the CCA to deliver the most favorable conditions for application QoE consistently. At a high level, Mortise first infers the application's desired QoS tradeoff from real-time QoE gradients, then adjusts CCA parameters to align with this target. To compute the exact parameter values, Mortise employs a control-theoretic analysis of system dynamics in rule-based CCAs, deriving closed-form mappings between parameters and their corresponding resulting QoS tradeoffs.

We implement Mortise on TCP and evaluate with both emulation and the CDN production environment at one of the world's largest Internet companies. Our real-world A/B testing across 128 cities worldwide over 3 months demonstrates that we can enhance the average QoE of our file downloading service by 11%-73% and the average QoE of video streaming services by 20%-167%, compared to other baselines in wireless scenarios, with negligible deployment overhead.

In summary, we make the following contributions:

- By demonstrating the mismatch between CCA and application, we identify the necessity for CCAs to continuously and quantitatively adjust their parameters to cater to network-dependent application preferences. (§2)
- We develop a user-friendly, deployable framework that extracts application preferences in real-time and dynamically adjusts the parameters of CCAs for them to optimize for QoE performance consistently. (§3)
- We implement (§4) and evaluate our framework with both trace-driven emulations and real-world production tests (§5), demonstrating its capability to adapt to application requirements at all times.

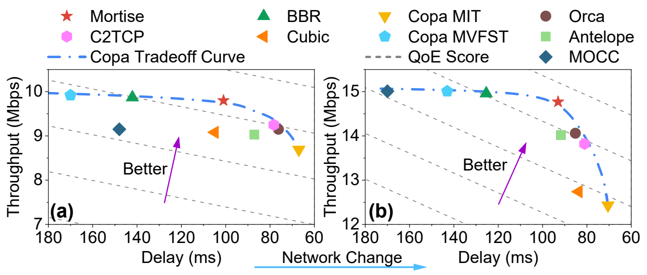


Figure 2: Operating points of different CCAs under different scenarios: (a) is stable, and (b) is more fluctuating.

2 Background & Motivation

Each congestion control algorithm (CCA) relies on a set of parameters to shape its control strategy, whether rule-based or optimization-based. For example, in BBR, specific parameters dictate the magnitude of probing, while in PCC, the weight factors in the objective function determine how different QoS metrics are balanced. Adjusting these parameters can fundamentally alter a CCA's behavior, enabling it to operate at entirely different points in the performance tradeoff space. For example, modifying the parameters of the delay-based CCA Copa can yield either a delay-sensitive variant (Copa MIT [7]) or a throughput-focused variant (Copa MVFST [18]), as illustrated in Fig. 2. This highlights the critical role of parameter configuration in determining CCA performance.

This work explores how to configure CCA parameters to achieve high QoE in dynamic networks consistently. §2.1 presents key observations that motivate the need for dynamic and network-aware parameter adjustment. §2.2 analyzes the limitations of existing solutions that rely on static and heuristic configuration. §2.3 introduces our core idea for enabling real-time, optimal parameter adaptation to approach the optimal operating point under dynamic network conditions.

2.1 Observation

Observation #1: CCAs face the tradeoff between throughput, delay, and loss. In theory, optimal CCA performance is achieved when the sending rate perfectly matches the available bandwidth while maintaining an empty network queue. This ideal state-known as Kleinrock's point [27]-maximizes throughput while minimizing both delay and loss.

Achieving Kleinrock's point consistently is practically impossible [24] due to unpredictable network fluctuations and inevitable control loop delays. To adapt to bandwidth changes, CCAs have to actively probe the network state by adjusting their sending rates. These probes help infer key states such as available bandwidth and propagation delay, which inform subsequent congestion control decisions.

However, probing is inherently uncertain. Increasing the sending rate might reveal unused bandwidth, or it could simply lead to increased queuing delay. This uncertainty leaves CCAs' decision-making process facing a dilemma between *conservative* and *aggressive* control strategies. While a conservative sending can maintain low latency and minimal loss, it tends to underutilize the available bandwidth and obtain lower throughput. Conversely, an aggressive sending could make fuller use of the available bandwidth but may suffer from longer queuing delays and even cause packet loss.

As a result, each CCA operates at a different operating point along the tradeoff spectrum, depending on its control strategy. We define this operating point as the resulting transmission performance, including throughput T, delay D, and loss L under the current network states, represented by $P(T,D,L)$. These operating points typically scatter across an area bounded by a Pareto frontier [44, 48, 59], as illustrated in Fig.2, indicating that no single CCA can outperform all others across all performance metrics. Any improvement in one dimension often comes at the expense of another.

Observation #2: The optimal operating point P^* depend on network state. The optimal operating point, denoted as $P^*(T,D,L)$, represents the best tradeoff among throughput, delay, and loss that maximizes QoE. This point lies on the Pareto frontier and is illustrated as the red star in Figure 1. Our key observation is that P^* is not fixed, but varies dynamically with the network state, which we define as the statistical characteristics (e.g., mean, variance, skewness) of network features such as available bandwidth, latency, and packet loss. We illustrate this with two examples.

The first example involves the level of network bandwidth.

- High bandwidth scenario: When the available bandwidth is high (e.g., resulting in a high bitrate for video streaming), further increasing throughput produces diminishing QoE gains, i.e., video quality metrics such as SSIM [19, 54] or VMAF [30] tend to saturate. In this case, a more conservative configuration that prioritizes lower latency and reduced packet loss can improve user experience more effectively than aggressively pushing for maximum throughput.
- Low bandwidth scenario: When bandwidth is limited, QoE becomes highly sensitive to throughput changes. Even modest increases in bitrate can lead to significant improvements in user-perceived video quality. In such cases, a more aggressive strategy that pushes the sending rate closer to the estimated available bandwidth is more beneficial.

This behavior arises from the nonlinear relationship between QoE and QoS. As shown in Fig.3, video quality improves sharply with bitrate at low rates but flattens out at higher rates. Thus, when throughput is already high, shifting focus to latency and loss control can enhance responsiveness with little impact on video quality. Conversely, under constrained throughput, improvements in bitrate have an important effect on user experience and should be prioritized.

Another example concerns bandwidth volatility:

- Stable network conditions: When bandwidth exhibits low variability, it is generally safe to increase the sending rate close to the estimated mean bandwidth without risking sudden queuing delays.
- Fluctuating network conditions: In highly variable networks, aggressive sending can lead to rapid queue build-up and latency spikes due to unexpected bandwidth drops. In this case, a conservative approach that leaves room for potential drops is crucial to prevent QoE degradation.

These examples illustrate that the optimal tradeoff among QoS metrics and thus the optimal operating point is inherently network-dependent, shaped by the nonlinear relationship between QoS and QoE under varying network conditions.

Observation#3: Network states experience substantial changes frequently within a session. Previous studies [17, 46] have shown that network states typically fluctuate on the time scale of tens of seconds. Our measurements from production environments and passive data analysis on datasets from Salsify [19] and Puffer [57] also confirm this observation (detailed distributions are presented in the Appendix A). We find that the duration of segments under the same network conditions mainly falls within the range of 5-60s, and significant changes, such as bandwidth changes exceeding 50%, often occur between segments. The frequent and substantial shifts in network conditions necessitate real-time identification and achievement of optimal operating points, which could potentially be located anywhere on the Pareto frontier.

Conclusion: *Static parameter settings are insufficient for maximizing QoE in dynamic network environments. To consistently achieve optimal performance, CCAs must adapt their parameters in real time to track the optimal operating point P^*, which varies dynamically with the evolving network state.*

2.2 Limitation of Existing Solutions

CCA design. Modern CCA can be broadly categorized into two types: rule-based and optimization-based. Rule-based CCAs (e.g., BBR [12], Copa [7]) rely on explicit network measurements to guide control actions, while optimization-based approaches (e.g., PCC [16], Aurora [25]) formulate congestion control as an objective maximization problem. In both cases, the design typically involves some form of network modeling to guide decision-making. Despite these advances, most CCAs still rely on static parameter configurations, which are ill-suited for dynamic and heterogeneous application requirements.

CCA adaptation approach. Different applications have varying QoE requirements: some applications prioritize low latency, while others prefer high throughput. To better accommodate such diverse needs, recent studies have introduced CCA adaptation frameworks. For example, Antelope [64], Floo [60], and Nimbus [21] can dynamically switch between CCAs or operational modes based on predefined objectives, enabling coarse-grained adaptation. Other approaches, such as MOCC [32], apply multi-objective reinforcement learning with transfer learning to gradually adjust optimization goals across different applications, though the adaptation typically takes several minutes. Additionally, methods like DeepCC [3] and C2TCP [1] expose tunable parameters to applications, allowing them to steer operating behavior more directly. While these methods offer configurable knobs that enable adaptation

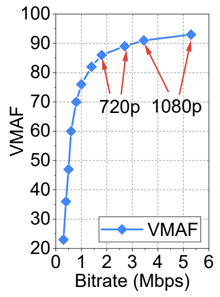


Figure 3: The relationship between bitrate and VMAF

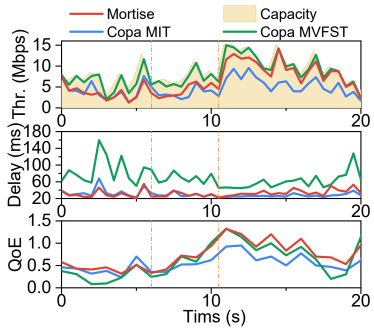


Figure 4: A trace to compare the performance of static and heuristic parameter configuration with dynamic and quantitative parameter configuration (Mortise)

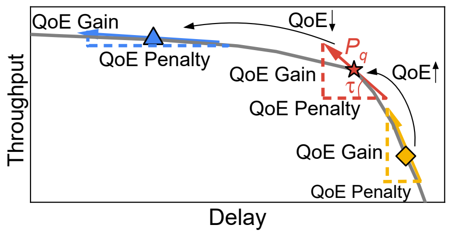


Figure 5: Illustration of how QoE changes with tradeoff changes

among QoS metrics, they fall short in guiding how to set these parameters to optimally maximize QoE. As a result, they still suffer from sub-optimal QoE.

However, whether through fixed CCAs or adaptive frameworks, existing solutions fall short in supporting continuous, fine-grained parameter tuning for optimal QoE in evolving network conditions. Their limitations are summarized below:

(1) Heuristic and static parameter configuration fail to maximize QoE. These solutions set parameter configurations by intuition or in a qualitative manner, and won't change them during a session. For instance, to serve video conferencing applications, they select delay-sensitive CCAs [7, 10] or assign a higher weight to the latency term in the objective function [32, 45]. Meanwhile, for the web browser applications, they switch to throughput-oriented CCAs [12] or assign a set of parameters to make CCAs probe more aggressively [18].

They all lack an accurate quantitative understanding of how parameters should be configured. The heuristic configurations can result in imprecise parameter settings in rule-based algorithms or imprecise weight factors in learning-based algorithms. Moreover, these configurations lack explicit guidance on how to make adjustments during a session, which leads existing solutions to use a fixed set of parameter configurations throughout a single session. As a result, whether a single CCA or an adaptive method, the operating points they achieve under diverse network conditions often deviate from the optimal points expected to maximize application QoE, as the two Copa variants displayed in Fig.4.

(2) They have limited flexibility and exhibit poor adaptability to changes in parameter configurations. As mentioned above, existing solutions largely depend on learning-based methods for adaptation, with a QoS optimization objective as their guiding principle. While transfer learning provides an intuitive and straightforward way to adjust inherent QoS objectives to achieve different operating points, it struggles to support frequent adaptations with low overhead during a session. On one hand, the transfer process can take several minutes or even hours [25, 32, 60], hindering the ability to swiftly adjust within a session to align with the rapid changes in network conditions. On the other hand, the resource overhead for training and performing transfer learning is substantial [2, 17], making it untenable in a production environment due to its high cost. Furthermore, CCA selection schemes are limited to a few discrete operating points, preventing them from maximizing arbitrary optimization objectives [32].

2.3 Basic Idea

To support frequent, flexible, and low-overhead adaptation, our framework builds on rule-based CCAs which incorporate explicit network models in decision-making, such as BBR [12] and Copa [7]. Compared to learning-based methods [32, 64], rule-based CCAs offer significantly higher computational efficiency, making them well-suited for real-time operation [2, 17]. Moreover, we observe that with careful parameter tuning, rule-based CCAs can achieve Pareto-optimal performance competitive with learning-based methods.

However, dynamically tuning these parameters to maintain QoE-optimal behavior under evolving network conditions consistently presents significant challenges:

- Modeling Complexity: The relationship between CCA parameters, network states, and QoE is highly complex, making it difficult to derive an analytical formulation. The joint parameter space is vast, making real-time optimization computationally expensive.
- Metric Mismatch: CCAs optimize for QoS metrics (e.g., throughput, latency, loss), whereas QoE depends on application-level semantics (e.g., SSIM, flow completion time).
- Feedback Latency: QoE metrics (e.g., video stall ratio) typically require extended observation periods and often only become available upon session completion [14]. This inherent delay makes QoE metrics poorly suited as feedback signals for real-time control systems, as they introduce both latency and ambiguity in the control loop.

To address these challenges, we introduce the QoS tradeoff as a proxy, rather than attempting to model QoE end-to-end. This tradeoff characterizes the proportional relationship between throughput, latency, and packet loss. Formally defined in §3.2, it quantifies how variations in throughput translate into corresponding changes in delay and loss. For each application, the preferred tradeoff expresses the maximum acceptable degradation in latency and packet loss for a given increase in throughput, ensuring QoE is preserved. By framing QoE optimization as a problem of dynamically adjusting to the preferred QoS tradeoff, we can leverage the native control knobs of rule-based CCAs (e.g., congestion window, target queue length), avoiding the need to directly map high-level QoE metrics (e.g., SSIM or stall ratio) to low-level parameters.

This tradeoff proxy is effective because operating at the optimal QoS tradeoff enables the CCA to consistently function at the most favorable operating point for application QoE. Intuitively, the tradeoff corresponds to the slope of the tangent line at the current operating point on the Pareto frontier, as illustrated in Fig.1. Given the convex nature of the Pareto frontier, tradeoff changes are monotonic. Let point P_q denote the operating point that aligns with the application's preferred QoS tradeoff. As illustrated in Fig.5, pursuing higher throughput than P_q (i.e., moving to points left of P_q) results in greater QoE penalties (from increased delay or packet loss) while yielding diminishing QoE gains from the additional throughput. As a result, these points provide lower overall QoE than P_q. Conversely, for points to the right of P_q, increasing throughput contributes more to QoE improvement than the associated penalties. Thus, these points also deliver lower QoE compared to P_q, as the application would benefit from moving closer to P_q. Therefore, P_q is the optimal operating point P^* that optimally balances throughput, delay, and loss to maximize application QoE.

Moreover, the tradeoff proxy simplifies a complex, high-dimensional control problem into a manageable scalar metric. For example, optimizing video QoE no longer requires modeling how Copa's queue length influences SSIM. Instead, it reduces to tracking the real-time relationship between throughput and delay. These tradeoffs can be computed from instantaneous QoS measurements (e.g., RTT gradients, throughput variance), bypassing the need for delayed feedback from application-level QoE metrics like stall ratio.

Thus, we use the QoS tradeoff as a bridge between application QoE requirements and CCA parameter tuning, as shown in Fig. 6. This approach decouples the optimization process into two layers: First, it derives the application's real-time preferred QoS tradeoff, which serves as a directional signal and target for control. Then, it adjusts the CCA parameters to steer the system toward an operating point that achieves this tradeoff, thereby maximizing QoE. This design preserves the simplicity and deployability of rule-based CCAs while enabling continuous, Pareto-optimal adaptation to dynamic application-level QoE demands.

3 Design

We first provide an overview of the design (§3.1) and then elaborate on our design components in §3.2 and §3.3.

3.1 Overview

For Mortise to achieve real-time optimal parameter adjustment, there are mainly two challenges:

(1) *How to extract the tradeoffs preferred by QoE under current situations?* The QoE model is dynamic and cannot simply be fed to the CCA at the start of session. On the one hand, QoE model may suggest very different preferences under different network conditions. On the other hand, QoE model would also be influenced by real-time non-network application factors (e.g., video resolution [33, 62], content [14, 53] or viewing context [62]). In response, we design an *application preference extractor* in §3.2 that uses the ratio of partial derivatives to dynamically extract the preferred QoS trade-off hints of various kinds of applications.

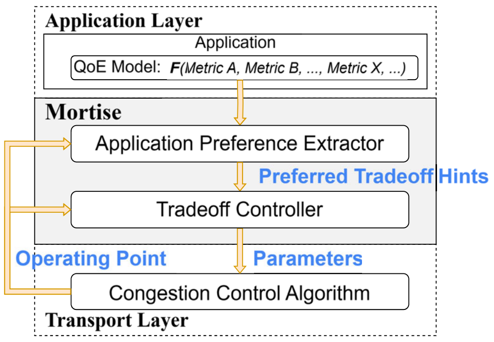


Figure 6: High-level block diagram of Mortise

(2) *How to identify the CCA parameter configuration to achieve the preferred tradeoff?* It is hard to directly determine the optimal CCA parameter, as it affects both the operating point and the preferred QoS tradeoff in a cross-dependent manner. Under the current operating point, when we tune the CCA towards the preferred QoS tradeoff, we would move to a new operating point, and the preferred QoS tradeoff at that point could also be different. In response, we design a *tradeoff controller* that *iteratively* tunes the CCA parameter based on the current QoS tradeoff in §3.3 and converges to the optimal CCA parameter in typically a few RTTs.

We present the overall workflow of Mortise in Fig.6. In essence, Mortise's application preference extractor obtains real-time preferred tradeoffs from the application's QoE model, taking into account the current operating point. The tradeoff controller then uses these tradeoff preferences as hints to adjust parameters, also based on the current operating point. Through iterative repetition of this process, Mortise is ultimately able to converge to the optimal operating point that delivers the best QoE.

3.2 Application Preference Extractor

We start by getting the QoE models of the application.

Getting Explicit QoE Models. An explicit QoE model would greatly help us calculate the application preference. Many applications already have well-established explicit QoE models developed through extensive research and production deployment experience, such as video streaming [19, 34]) and web browsing [42, 60]. Some have implicit QoE models (e.g., Markov-chain [14] or learning-based [45]), which could also be approximated by explicit ones (e.g., converting them to decision trees [36]). In such cases, Mortise's only requirements for application integration are that they provide their QoE models at connection initialization and inform again when application factors change.

Extract the Preference Hints. With explicit QoE models, we now extract the preferred tradeoff. Inspired by local linearization, we observe that the *ratio of partial derivatives* (we denote as the *hint*) could well reflect the preferred QoS tradeoff. Take the ratio of the partial derivative of latency to that of throughput as an example; it represents the maximum latency increase that an application can tolerate (and feel worthwhile) to increase its throughput. Also, we observe that most QoE models are differentiable or only with discontinuities of the first kind [28, 29], making derivatives practical.

We now present the mathematical expression. we represent the QoE model as a function F composed of n metrics $\mathcal{M}_i$:

$$QoE = F(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_n) \tag{1}$$

Given that each metric $\mathcal{M}_i$ is directly associated with throughput, delay, or loss, we can express the QoE as its corresponding local QoS format, a function f on throughput T, delay D, and loss L:

$$QoS = f(T, D, L) \tag{2}$$

Assuming the current operating point is $P_c(T_c, D_c, L_c)$, where throughput, delay, and packet loss are T_c, D_c, and L_c respectively, we can calculate the partial derivatives in each direction by solving the gradient:

$$\nabla f = \left(\frac{\partial f}{\partial T}, \frac{\partial f}{\partial D}, \frac{\partial f}{\partial L}\right)_{(T_c, D_c, L_c)}$$

Hence, we can calculate the two application preferred tradeoff hints λ (for delay-throughput tradeoff) and β (for loss-throughput tradeoff) by the ratio of partial derivatives as:

$$\begin{aligned}\lambda &= \left| \left(\frac{\partial f}{\partial D}\right)_{(T_c, D_c, L_c)} \Big/ \left(\frac{\partial f}{\partial T}\right)_{(T_c, D_c, L_c)} \right| \\ \beta &= \left| \left(\frac{\partial f}{\partial L}\right)_{(T_c, D_c, L_c)} \Big/ \left(\frac{\partial f}{\partial T}\right)_{(T_c, D_c, L_c)} \right|\end{aligned} \tag{3}$$

Extend to Statistical Metrics. In practice, applications may employ statistical metrics, such as average-based [60] or distribution-based (jitter, percentile, etc.) [37, 38]. The aforementioned form remains applicable for metrics, such as average-based metrics, which can be expressed as a linear combination of some simple QoE metrics. However, for distribution-based metrics, this form falls short of accurately capturing such situations. Consequently, we extend T, D, L to vectors $\mathbb{T}, \mathbb{D}, \mathbb{L}$ to symbolize the distribution of throughput, delay, and loss, respectively. Similarly, we can derive the extended application preference hint λ_{ext} and β_{ext}:

$$\begin{aligned}\lambda_{ext} &= \left| \Delta f(\vec{\mathbb{T}}_c, \vec{\mathbb{D}}_c + \Delta\vec{\mathbb{D}}, \vec{\mathbb{L}}_c) / \Delta f(\vec{\mathbb{T}}_c + \Delta\vec{\mathbb{T}}, \vec{\mathbb{D}}_c, \vec{\mathbb{L}}_c) \right| \\ \beta_{ext} &= \left| \Delta f(\vec{\mathbb{T}}_c, \vec{\mathbb{D}}_c, \vec{\mathbb{L}}_c + \Delta\vec{\mathbb{L}}) / \Delta f(\vec{\mathbb{T}}_c + \Delta\vec{\mathbb{T}}, \vec{\mathbb{D}}_c, \vec{\mathbb{L}}_c) \right|\end{aligned} \tag{4}$$

If none of the metrics take into account the distribution of throughput, latency, or loss, then λ_{ext} and β_{ext} naturally reduce to the λ and β in Eq.3. For details on how the extractor works on the specific QoE models of real-world applications, we refer the readers to §4, Appendix C.2 and Appendix D.2.

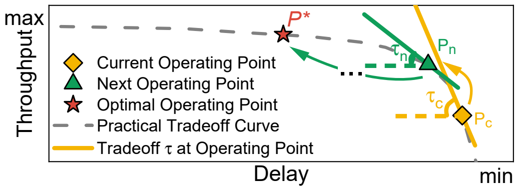


Figure 7: An illustration of how tradeoff controller works. P and τ denote the operating point and the latency-throughput tradeoff

From our online observations, the extracted hints can basically correspond to the actual application preferences and user experiences, providing valuable guidance for adjusting CCA strategies. We leave empirical analysis on passive datasets [57] for more comprehensive validation in the future.

3.3 Tradeoff Controller

As mentioned in §3.1, we could not directly solve the optimal parameters for unknown network conditions. To address this challenge and inspired by coordinate descent [55], we adopt a balancing feedback control mechanism with an iterative two-step strategy. We first measure the current operating point and the delay-throughput and loss-throughput tradeoffs it achieves. We then obtain the preferred tradeoffs (the hints) that the application aims to reach at this operating point from the extractor. Based on the discrepancy between them and the current tradeoffs, we identify the next tradeoffs to align more closely with the desired preference. Subsequently, we estimate the parameters to achieve the tradeoffs and set them to the CCA, pushing the CCA to a new operating point. Following this, we re-estimate the new preferred tradeoffs at the new point and continue this cycle of balancing feedback control. This approach enables our estimation of application preferences (i.e., hints) to progressively approach the actual preferred tradeoff at the optimal point, allowing us to continually reduce the discrepancy and converge to the optimal operating point that delivers maximum QoE.

Measure Tradeoff of Current Operating Point. Initially, we need to measure the tradeoffs at the current operating point P_c (represented by the yellow diamond in Fig.7), such as the delay-throughput tradeoff τ_c indicated by the slope of the yellow dashed tangent line in Fig.7, to compare them with the application's preferences hints. By measuring the throughput, delay, and packet loss at the current and *nearby* operating points, we can approximate the tradeoffs (the tangent lines) at the current operating point using a secant method.

In production environments, measurement noise and network fluctuations always hinder us from obtaining accurate results in a short period of time. In other words, the longer we measure, the more accurate the results we can obtain. Thus, we adapt the time period for measuring the tradeoffs. When we are still far from the optimal point, rough results are sufficient to guide the direction, so we measure fewer rounds to save time. As we approach the optimal point, we measure for longer periods to obtain more accurate results for fine-tuning.

The Balancing Feedback Control. Once we have measured the tradeoffs at the current operating point and extracted the real-time application preference hints λ and β at this point, we can proportionally adjust the CCA's tradeoff based on their discrepancy. We set the adjustment step length as a proportional factor σ multiplied by the discrepancy. With the tradeoff τ_c at the current operating point, the tradeoff τ_n of the next operating point P_n (represented by the green dashed line at the green triangle in Fig.7) can be calculated as follows:

$$\tau_n = \sigma\lambda + (1-\sigma)\tau_c \tag{5}$$

Following the Averaging Principle [6], we default σ to 0.5, aiming to strike a balance between rapid reaction and stability. Given the inaccuracies in measuring the tradeoff of the operating point, we limit the frequency of adjustments when the discrepancy with the preferred tradeoff is sufficiently slight, thereby avoiding oscillations. Still, the controller persistently monitors the tradeoffs, preparing for adjustments in response to changes in network conditions or application preferences.

The Parameters to Adjust. After determining the next tradeoff, we need to achieve it by adjusting the parameters. We observe that modern rule-based CCAs [7,12] mainly influence the CCA's operating point and tradeoffs by influencing either steady-state queue length (i.e., equilibrium point) or the transient probing speed (i.e., path to converge to the equilibrium point). For different CCAs, the parameters influencing these two factors are those that Mortise can adjust. Also, as illustrated by the blue dashed curve in Fig.2, the possible tradeoff curve obtained by adjusting these parameters can approximate the Pareto frontier. While we acknowledge the influence of other parameters, these two types are more straightforward to comprehend and control. Thus, this work primarily focuses on them. While the accurate mapping between parameters and QoS tradeoffs is initially unknown under new network conditions, rough parameter adjustments that match the direction guided by the hints are sufficient. As the historical data accumulated through iterative adjustments, the panorama of the mapping would be clearer, allowing us to make accurate decisions for desired QoS tradeoffs. For the detailed process of how the feedback control advances to convergence, we refer the readers to §5.2.

Change-Point Detection. The feedback control is adequately equipped to achieve rapid convergence in common scenarios. However, for certain transient scenarios, such as drastic bandwidth degradation, worsening network conditions can delay the control loop for tradeoff measurements and feedback acquisition. Additionally, historical data can interfere with the controller. Together, they prevent the CCA from converging to the new optimal operating point on time. Hence, we introduce an online change-point detection (CPD) mechanism [4] on bandwidth. This mechanism prompts the controller to discard historical records and initiate a broad exploration under new network conditions, facilitating faster convergence. Although the CPD may yield false positives due to outliers, it only triggers a new exploration, with the primary adjustments still governed by the balancing feedback control. In other words, incorrect detection exerts minimal impact on performance.

4 Implementation & Experiment Setup

Underlying CCA and stack. We implement Mortise[1] on Copa, since it has a clear network model, a simple internal mechanism, and an explicit parameter δ that directly influences both the steady-state queue length and the probing speed. Our implementation builds on Linux kernel TCP with eBPF (Extended Berkeley Packet Filter) [31, 35], eliminating the need for kernel modifications. Although we focus on Copa for detailed analysis, Mortise is a general-purpose framework whose design principles can naturally extend to other CCAs. Additional prototypes on GCC for WebRTC [43] (Appendix C) and Copa for QUIC [51] (Appendix D) demonstrate over 10% improvements in both cases, validating Mortise's generality. We leave further exploration for future work.

Low system overhead and ease of Deployment. Our user-space controller exchanges data with the in-kernel CCA via *ebpf_map*, which functions as shared memory. The CCA uses this map to report the measured information of the current operating point, while the controller uses it to send the parameters to the CCA. Given that our control and communication occur at the RTT level rather than on a per-packet basis, the data can be aggregated and communicated at low frequency. This asynchronous communication method fully meets our requirements and incurs minimal communication overhead. The main part of the framework implemented for Linux kernel TCP can be reused across stacks and applications (for QUIC, WebRTC). Additionally, our alterations to the CCA are minimal. In the case of the 1000-LoC (line of code) Copa, the feature of applying parameters requires a mere 15 LoC, and reporting network measurements only 20 LoC.

Applications and their QoE model settings. We evaluate the CCAs under two types of applications: (1) a file downloading application (including web browsing), where the client sends a series of requests and the server provides responses accordingly. (2) a streaming video-on-demand application, where the client employs the VLC Adaptive Bitrate (ABR) algorithm [41], a method combinedly based on current playback buffer and throughput estimation, to download video chunks from the server. We set up different QoE models for them based on the characteristics of the applications and empirical values in the production environment as follows:

(1) File Downloading. For file downloading applications, the main concerns are how fast the file request is completed (reflected in the normalized RCT) and traffic costs for the CDN to complete the request (reflected in the retransmission ratio). Thus, the QoE model is set to be:

$$\mathrm{QoE} = \frac{\mathrm{R}}{RCT}(1-\mathrm{P}(r)) \tag{6}$$

[1]Open sourced at: https://github.com/BobAnkh/Mortise

where R is the request size, r is the retransmission ratio, RCT is the request completion time, and $P(r)$ is a penalty function for retransmissions:

$$\mathrm{P}(r) = ar + b,\ (a,b) = \begin{cases} (0,0), & 0 \leq r < 0.05 \\ (4,-0.2), & 0.05 \leq r < 0.1 \\ (1,0.1), & 0.1 \leq r < 0.4 \\ (0,0.5), & 0.4 \leq r \leq 1 \end{cases}$$

The intuition behind this piecewise function is that a few retransmissions will not impact QoE, while the penalty on QoE becomes progressively more severe with increasing volume of retransmissions. We further translate the QoE model to be a function of QoS metrics (i.e., throughput T, delay D, packet loss rate L) to get the QoS formula:

$$QoS = f(T,D,L) = \frac{R[1-(aL+b)]}{R/T+D+DL/2}$$

We can further extract the application preference hints λ and β given current operating point $P_c(T_c,D_c,L_c)$:

$$\begin{aligned} \lambda &= \left|\frac{\partial f}{\partial D} \Big/ \frac{\partial f}{\partial T}\right| = \frac{T_c^2(L_c+2)}{2R} \\ \beta &= \left|\frac{\partial f}{\partial L} \Big/ \frac{\partial f}{\partial T}\right| = \frac{T_c[2a(R+T_cD_c)-(b-1)T_cD_c]}{2R(1-aL_c-b)} \end{aligned} \tag{7}$$

QoS tradeoffs are not only influenced by current network conditions but also by the size of the requested content.

(2) Video Streaming. The QoE of video streaming applications is mainly decided by the video quality (reflected in the bitrate) and the stall events (reflected in the stall time) [34, 47]:

$$\mathrm{QoE} = \sum_{n=1}^{N} q(B_n) - \rho \sum_{n=1}^{N} T_n \tag{8}$$

for a video with N chunks. B_n represents the bitrate of $chunk_n$ and $q(B_n)$ maps that bitrate to the video quality perceived by a user. T_n represents the stall time that results from downloading $chunk_n$ at bitrate B_n, and ρ is a weight factor to balance between them (we empirically set $\rho = 2.66$).

q is set as a logarithmic function, $q(B_n) = \log(B_n/B_{min})$, where B_{min} is the minimal bitrate (400 kbps in our app). The intuition is that bitrate increase would be marginal to the perceived quality when it is already high. Stall occurs when the download time exceeds the remaining playback buffer; thus, it can be modeled as a piecewise function. Hence, we can get the empirical QoS formula:

$$QoS = f(T,D,L) = \log(\frac{T}{B_{min}}) - \rho \cdot \max(0, \frac{C}{T} + D(1+L) - V)$$

C is the chunk size and V is the length of the current playback buffer. We can also extract the preferred QoS tradeoff hints:

$$(\lambda,\ \beta) = \begin{cases} \left(\frac{\rho T_0^2(1+L_0)}{T_0+\rho C},\ \frac{\rho T_0^2 D_0}{T_0+\rho C}\right) & \text{if } C/T_0 + D_0(1+L_0) > V \\ (0,\ 0) & \text{otherwise} \end{cases} \tag{9}$$

The QoS tradeoff is influenced by current network conditions, ABR decisions, video slicing strategies, and user playback buffering policies.

Baselines. To evaluate the performance and effectiveness of Mortise, we compare it with existing solutions as follows:

- Throughput-oriented CCA:
  1. Cubic [22]: a classic loss-based heuristic CCA.
  2. BBR [12]: a CCA controls the rate with an explicit model that estimates available bandwidth and RTT.
- Delay-based CCA:
  3. Vegas [10]: a classic delay-based heuristic CCA.
  4. Copa MIT [7] and Copa MVFST [18]: two Copa variants adopting $\delta = 0.5$ for low latency and $\delta = 0.04$ for high throughput, respectively.
- Learning-based CCA:
  5. Orca [2]: a single-objective RL-based CCA combined with the rule-based CCA (Cubic).
- CCA with Adaptability:
  6. MOCC [32]: a multi-objective RL-based CCA with transfer learning to migrate to new objectives.
  7. Antelope [64]: a CCA selection method on Cubic, BBR, Copa[2], etc.
  8. Antelope Copa: a CCA selection method on five different Copa profiles (different δ values).
  9. C2TCP [1]: a flexible cellular CCA with an interface to set desired average target delay requirements.
  10. DeepCC [3]: an RL-based cellular CCA with an interface to set desired average target delay requirements.
- Our Solution:
  11. Mortise Disc. we tweak our complete implementation to select from only five discrete δ values for adjustment.
  12. Mortise, our complete implementation which can assign arbitrary δ to the underlying Copa, based on requirements.

To ensure a fair comparison, we retrain or fine-tune each learning-based CCA for different applications.

Real-World A/B Testing. We have deployed Mortise for A/B testing in our CDN production environment in one of the largest companies to serve real applications (file downloading and video streaming) across the Internet with real cross traffic and packet schedulers. All modifications are on server-side, requiring no client-side changes. We evaluate in both wired and wireless scenarios. For each incoming connection, we randomly assign one of the baselines to handle all traffic on that connection. To encompass a wide range of real-world situations, our A/B test covered 128 cities all over the world and extended over more than 3 months.

Emulation. We also conduct offline experiments[3] via Mahimahi [42] for detailed analysis with wireless traces from Orca [2], including 3 common scenarios (walking, driving, and stationary). We set the minimum RTT to 20ms and the buffer size to 150KB. We let each CCA operate under real applications to send traffic over these traces, repeating the tests 5 times. For file downloading, we replay real workloads captured from production. The characteristics of the workload, i.e., request interval and response size, are left in Appendix B.

[2]We retrain it to include Copa as a candidate CCA for fair comparison
[3]Emulate on a Dell R740 server with 80 CPU cores, 384GB RAM.

5 Evaluation

We evaluate Mortise with emulation and real-world tests from various aspects, including performance (§5.1 and deep-dive understanding in §5.2), overhead (§5.3), parameter sensitivity (§5.4), and fairness and friendliness (§5.5).

5.1 Performance

We present that Mortise consistently achieves superior QoE performance over various network scenarios for different applications. Fig.8 shows the performance distribution of all algorithms tested across two applications and three network scenarios. For file downloading, we display the distribution of request completion time (RCT), retransmission ratio, and overall QoE; for video streaming, we illustrate the distribution of bitrate, stall ratio, and overall QoE.

Main Takeaway: Existing solutions may perform well for certain application or network scenarios, but perform poorly in others. In contrast, Mortise consistently provides the most suitable operating points for different applications across various network scenarios, significantly outperforming other baselines. It delivers optimal performance across all cases, boosting the QoE for file downloading by up to 160% and the QoE for video streaming by up to 167%.

Remark 1 (Mortise): We are the only method that consistently achieves superior performance in QoE and all its constituent metrics, across all application types and network scenarios. The only exception pertains to the retransmission ratio of file downloading in wired scenarios, as depicted in Fig.8c. Our retransmission ratio of 0.14% is a mere 0.09% higher than the best one, and such a small ratio is already imperceptible in a production environment. For fluctuating real-world wireless and emulated cellular scenarios, Mortise demonstrates substantial enhancements, elevating the average QoE for file downloading by 11.3%-73.5% (Fig.8a) and 10.8%-160.8% (Fig.8e), and for video streaming by 19.8%-167.2% (Fig.8b) and 11.4%-32.4% (Fig.8f). In contrast, in the more stable wired scenarios, the improvements are slightly smaller, with increases in the average QoE for the two applications by 7.2%-65.3% (Fig.8c) and 3.7%-11.6% (Fig.8d). Compared to baselines, we can especially optimize tail cases a lot by preventing severe degradation of any particular application metric with explicit preference as guidance. Furthermore, Mortise, with its capability to continuously adjust parameters, demonstrates a performance improvement of up to 16% over Mortise Disc, the variant limited to a few discrete parameter values. This underscores the necessity of continuous parameter tuning.

Remark 2 (Delay-Based): Typically, delay-sensitive CCAs (Copa MIT, Vegas) have a higher RCT in file downloading while causing less retransmission, due to their small queuing tolerance. In video streaming, they also get lower bitrates and higher stall ratios. Increasing the tolerated queue length (Copa MVFST) can achieve greater throughput, but it comes at the expense of a higher retransmission ratio.

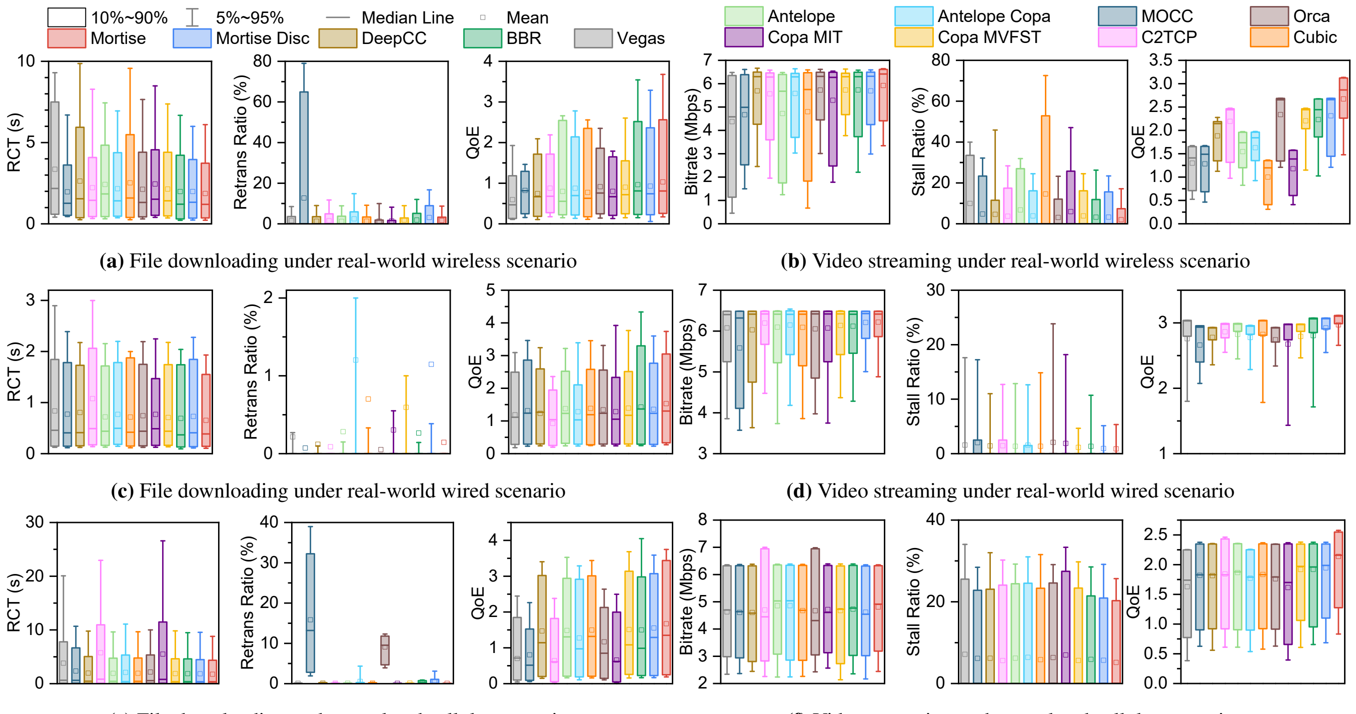
Remark 3 (Throughput-Oriented): In real-world wireless scenarios, Cubic can only achieve relatively low throughput due to its sensitivity to packet loss, a common occurrence in our wireless production environment. It exhibits acceptable performance in wired and cellular scenarios where packet loss is less. BBR provides good throughput support for file downloading with higher retransmission ratios. However, as it does not optimize for QoE, all of its metrics fall short compared to Mortise, with its overall QoE trailing ours by 7-11%. In the context of video streaming, BBR's performance deteriorates significantly, exhibiting greater variability and instability, and lags behind our solution by as much as 20%.

Remark 4 (Learning-Based): Orca delivers good performance in real-world wireless environments, but its performance in other cases is relatively unremarkable. For instance, its average QoE for video streaming applications in wired scenarios is low due to its higher stall ratio. Similarly, in cellular environments, the QoE for file downloading is also suboptimal due to a significantly higher retransmission ratio.

Remark 5 (CCA with Adaptability): While MOCC manages to achieve a low RCT in file downloading, it results in an extremely high retransmission ratio (Fig.8a, 8e), thereby harming its overall QoE. This could be attributed to its design as a fully learning-based method, which may fail to adequately generalize [2] to unseen network environments, resulting in overly aggressive sending behaviors. While the QoE offered by DeepCC or C2TCP is generally suboptimal, DeepCC exhibits slightly better performance under file downloading applications than under video streaming applications, whereas C2TCP performs better under video streaming applications than file downloading applications. For CCA selection methods like Antelope and Antelope Copa, their performance can be suboptimal and unstable due to the constraints of discrete options and the potential for incorrect choices. This can manifest as a high retransmission ratio (Fig.8a) or stall ratio (Fig.8b) under wireless scenarios. In conclusion, while offering adaptive interfaces, they lack the necessary mechanisms to adjust parameters in accordance with application preferences and network conditions, resulting in suboptimal performance.

5.2 Under the Hood

We now show the details of how Mortise functions. As shown in Fig.9a, the link capacity increases substantially around the 20-second mark, accompanied by a significant increase in fluctuations. Fig.9b and Fig.9c illustrate the temporal evolution of our chosen parameter δ, the application preference hint λ, and the current delay-throughput tradeoff around this 20-second mark. As β in this case is relatively small, we omit it in the figures. We also derive the optimal δ with offline exhaustive search (i.e., the Oracle in Fig.9d, 9e). As we observe, prior to the 20-second mark, Mortise has already converged near the optimal operating point, and δ only makes periodic minor adjustments around its optimal value. When the link capacity changes, with the underlying CCA increasing the sending rate, the operating point shifts, as well as the currently



(a) File downloading under real-world wireless scenario **(b)** Video streaming under real-world wireless scenario

(c) File downloading under real-world wired scenario **(d)** Video streaming under real-world wired scenario

(e) File downloading under emulated cellular scenario **(f)** Video streaming under emulated cellular scenario

Figure 8: Performance distribution of 2 applications (file downloading and video streaming) under 3 network scenarios (wired, wireless, and cellular). For a better QoE, file downloading strives for a lower RCT and a lower retransmission ratio, whereas video streaming aims for a higher bitrate and a lower stall ratio.

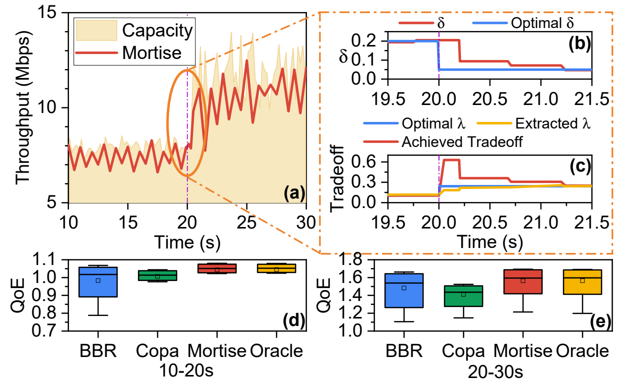


Figure 9: Details of the how Mortise functions under network conditions change

achieved tradeoff. The extracted delay-throughput preference hint λ rises (yellow line in Fig.9c), as the application tends to prioritize delay when the bandwidth gets high. However, the current δ value remains high, leaving the achieved tradeoff to deviate from the optimal. Guided by the hint, we need two additional rounds of parameter adjustments to converge, all within 1 second. This process is significantly shorter than the timescale of network changes, ensuring consistent tracking of optimal operating points without oscillation.

We also display the distribution of QoE between the 10s-20s and 20s-30s intervals in Fig.9d and Fig.9e, respectively, compared with two more baselines, BBR and Copa. Our deviation from the Oracle is minimal, with the slight discrepancy at the tail arising from the convergence process after network

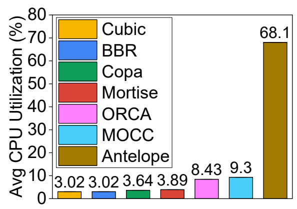


Figure 10: CPU overhead

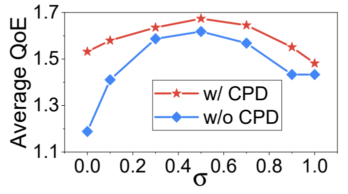


Figure 11: Impact of proportional factor σ and Change-Point Detection

conditions change. In contrast, algorithms like BBR or Copa, which use fixed and heuristically set parameters, either act too aggressively or too conservatively under both network conditions, delivering suboptimal QoE performance.

5.3 Overhead

To investigate Mortise's system overhead, we compare it with various state-of-the-art CCAs. We send traffic from a server to a client over an emulated channel (with 48Mbps bottleneck link and 20ms RTT) for 180 seconds and measure the average CPU utilization of these algorithms on the sender side. To have a fair comparison and reduce the impact of initialization phases required by some of these CCAs, we exclude the first few seconds for all CCAs. As shown in Fig.10, Mortise achieves very low overhead of less than 4% in total with only 0.25% additional overhead introduced on Copa, which is similar to two well-optimized in-kernel CCAs (i.e., Cubic and BBR). Its overhead is much lower than that of learning-based schemes, Antelope, Orca, and MOCC. The process of extracting preference hints simply involves calculating partial derivatives. To avoid redundant computations, we keep a record of all models computed during the session. Given the low frequency of extraction (at a scale of several seconds), the overhead of this process is almost negligible.

5.4 Parameter Sensitivity

We then evaluate the sensitivity of parameters in Mortise. We tune the adjustment step length proportional factor σ of the tradeoff controller from 0 to 1 with the file downloading app. As depicted in Fig.11, Mortise can still maintain high performance across a wide parameter range, striking an effective balance between responsiveness and stability. Given the potential inaccuracies in operating point measurements, large proportional factors σ could lead to recurrent oscillations. Conversely, smaller proportional factors could also impede convergence speed. Both will hinder the rapid convergence to the optimal operating point, thereby diminishing performance.

We also evaluate the impact of the change-point detection mechanism on performance. As observed, acceptable performance can still be achieved even in the absence of this mechanism, outperforming other CCAs. However, relying solely on the balancing feedback control's inherent adjustments may not be timely enough in certain scenarios, leading to potential performance losses. In essence, change point detection can facilitate a quicker adjustment.

We also evaluate the impact of buffer size on the performance of the file downloading application under the emulated cellular scenario. We vary the buffer size from $0.5\times$BDP to $16\times$BDP and present the performance in Fig.12. We observe that smaller buffers lead to more frequent packet loss and retransmissions, resulting in a performance decline across all methods, with aggressive algorithms like BBR experiencing a more pronounced drop. However, Mortise, due to its ability to dynamically adjust its operating point based on current network conditions to align with application preferences, can consistently maintain optimal and more stable performance.

5.5 Fairness & Friendliness

Fairness. We first evaluate how Mortise behaves in the presence of other Mortise flows. We run Mahimahi [42] emulations that let four flows using the same CCA compete for a bottleneck link with 48Mbps bandwidth, 20ms propagation delay, and $1\times$BDP buffer. The flows start one by one with 25s intervals. We use file downloading traffic in this subsection to evaluate fairness when all flows have enough data to send. We repeat the tests 5 times. Fig.13 shows the average throughput of flows for each CCA through time. As observed, Mortise shares bandwidth fairly between competing flows. It exhibits slightly higher fluctuations after convergence than Copa, as we will further explore for the optimal parameter.

We then evaluate the fairness of Mortise variants tailored for different applications. We select two variants: Mortise File and Mortise Streaming, using QoE models for file downloading and video streaming, respectively. We sequentially start four flows, either the variants or Cubic, on the same link, and depict the average throughput of different flows in Fig.14. As observed, Mortise File is slightly more aggressive than the other two. Mortise Streaming achieves comparable throughput when competing with Cubic, while smartly being more aggressive when competing with Mortise File. The intuition is that conflicting application performance preferences might be impossible to fully satisfy in certain cases, e.g., it is hard for Mortise Streaming to achieve low latency when competing with a buffer-filling CCA. In such case, Mortise would seek a reasonable point that provides the best possible service quality. When achieving low latency is a lost cause, we use a more aggressive strategy to prevent the flow from starvation.

Friendliness. We evaluate the friendliness of Mortise with TCP CCA. We first let a flow of different CCAs compete with a Cubic flow over a 48Mbps bandwidth and $1\times$BDP buffer link simultaneously. We vary the RTT from 20ms to 300ms and report the friendliness ratio defined by $\frac{\text{delivery rate of CCA flow}}{\text{delivery rate of Cubic flow}}$ in Fig.15. The results indicate that Mortise File is slightly more aggressive in obtaining bandwidth and Mortise Streaming achieves good TCP friendliness with Cubic. In Fig.16, we illustrate the throughput of the tested CCAs when they are competing with different numbers of Cubic flows. The RTT is fixed to 20ms. Two variants of Mortise all decrease their share when the number of competing Cubic flows increases, while MOCC does not yield bandwidth to more competing flows. In general, Mortise achieves good TCP friendliness.

6 Discussion

Network-Assisted CCAs. Algorithms like ABC [20], with rich feedback from in-network devices, could potentially achieve better operating points (higher throughput and lower latency) than the CCAs used in our paper. However, they still have to face the tradeoffs between QoS metrics and lack the capability to adjust parameters to match real-time application preferences. This suggests that our framework can still be applied on top of these CCAs to enhance application performance in such scenarios.

Cooperation with learning-based CCAs. Even though learning-based CCAs are comparatively complex, they still possess certain hyperparameters that can modulate the control process, such as the confidence amplifier $m(\tau)$ and dynamic change boundary ω in Vivace [16]. Our framework can still be applied to them, merely requiring more intricate mapping for these hyperparameters. This would also enable the adjustment of their operating points, facilitating better adaptation to application preferences. We leave the combination of this framework with these CCAs for future work.

Short-lived flows. For short-lived flows and other application-limited cases, the CCA is not the dominant factor in their performance. Implementing a better scheduling scheme at the host or in the network would significantly improve their performance, which is orthogonal to Mortise's design.

Ethical Consideration. We have obtained user consent for our A/B testing on the CDN involving real users. We also restrict how often each user is selected for the suboptimal

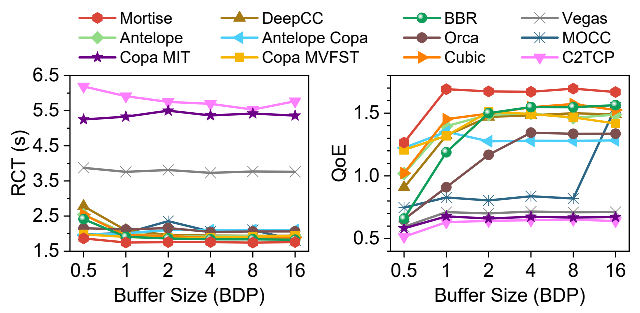


Figure 12: Impact of buffer size on performance

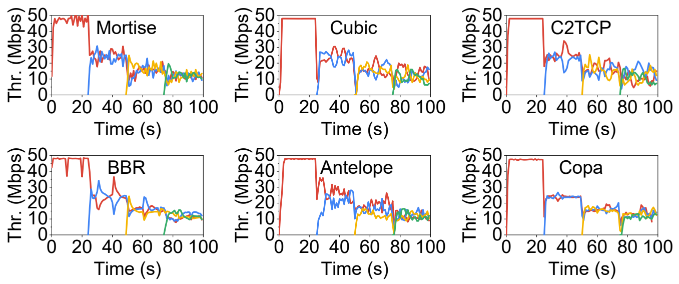


Figure 13: Throughput dynamics of different flows competing on one link for various CCAs

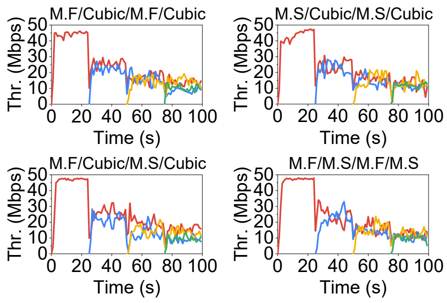


Figure 14: Throughput of competing flows of different Mortise variants (as noted above subfigures). M.F is short for Mortise File and M.S is short for Mortise Streaming

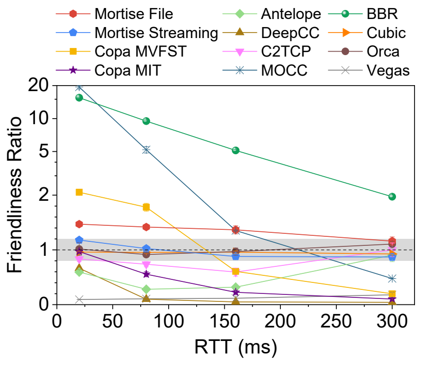


Figure 15: Friendliness Ratio of CCAs across different RTTs

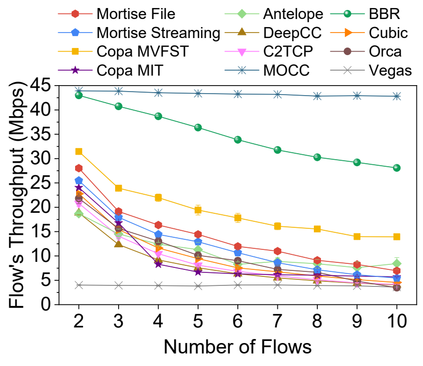


Figure 16: Flow's throughput across different numbers of competing flows

baselines to ensure that A/B testing does not significantly degrade the overall user experience.

7 Related Works

CCA-selection Methods. These approaches [58, 60, 64] typically choose several latency-sensitive and TCP-competitive CCAs to form a candidate pool and use learning-based techniques to select the best possible CCA periodically. However, these limited choices cannot accommodate arbitrary and network-dependent application preferences. Furthermore, since different CCAs have unique internal states, additional time is required to slow-start, probe from scratch, and converge [58, 64]. This incurs significant overhead when switching CCAs during runtime and can potentially cause up to 14% extra packet loss after the switch [60]. Precisely mapping the internal states across different CCAs is hard as their structures and physical meanings could be largely different [60].

Multi-Objective Learning-based Congestion Control. Building upon Aurora [25], MOCC [32] can migrate its QoS objective through transfer learning, facilitating adaptation for new applications. However, the migration process takes several hundred seconds [32], making it unable to flexibly align its tradeoff with application preferences and network conditions during a session. Additionally, such fully learning-based CCAs face issues of insufficient generalization, incorrect or slow convergence, and significant overhead [2].

eBPF (Extended Berkeley Packet Filter). eBPF is a highly flexible and efficient virtual machine-like construct in the Linux kernel, which allows for running sandboxed programs in the kernel without changing source code or loading modules. It has been employed in several innovative ways: optimize TCP parameters in datacenters [9], enhance TCP extensibility [52], etc. The mature eBPF technology in the kernel has enabled us to deploy Mortise without modifying the kernel. With the rapid development of user-space eBPF technology [23, 63], we are confident that in the near future, we will be able to implement our framework in user-space network stacks without requiring intrusive modifications.

8 Conclusion

In this paper, we propose Mortise to continuously and quantitatively adjust rule-based CCA's parameters to always align with application preferences for optimal QoE. Mortise introduces an extractor to acquire the application's preferred tradeoff in real-time and a tradeoff controller to adjust the corresponding parameters for that tradeoff. We further deploy the framework and find it well-performed in emulation and production. We believe it is vital to let CCAs gain fine-grained adaptability to the ever-evolving network applications.

Acknowledgements. We sincerely thank our shepherd, Stefan Schmid, anonymous reviewers, and labmates in the Routing Group from Tsinghua University for their valuable feedback. This work is supported in part by the National Natural Science Foundation of China (No. 62372261 and 62221003), in part by the Young Elite Scientists Sponsorship Program by CAST (No. 2022QNRC001), in part by the National Key R&D Program of China (No. 2025YFE0201000), in part by the Hong Kong RGC Early Career Scheme (No. 26212525), and in part by the Tsinghua-Bytedance Collaborative Grant. Bo Wang and Mingwei Xu are the corresponding authors.

References

[1] Soheil Abbasloo, Yang Xu, and H Jonathan Chao. C2tcp: A flexible cellular tcp to meet stringent delay requirements. *IEEE Journal on Selected Areas in Communications*, 37(4):918–932, 2019.

[2] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. 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*, pages 632–647, 2020.

[3] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Wanna make your tcp scheme great for cellular networks? let machines do it for you! *IEEE Journal on Selected Areas in Communications*, 39(1):265–279, 2020.

[4] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[5] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[6] Mohammad Alizadeh, Abdul Kabbani, Berk Atikoglu, and Balaji Prabhakar. Stability analysis of qcn: the averaging principle. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 49–60, 2011.

[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] Juraj Bienik, Miroslav Uhrina, Lukas Sevcik, and Anna Holesova. Impact of packet loss rate on quality of compressed high resolution videos. *Sensors*, 23(5):2744, 2023.

[9] Lawrence Brakmo. Tcp-bpf: Programmatically tuning tcp behavior through bpf. In *NetDev 2.2*. 2017.

[10] Lawrence S Brakmo, Sean W O'malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, 1994.

[11] Neal Cardwell, Y Cheng, K Yang, D Morley, SY Hassas, P Jha, Y Seung, V Jacobson, I Swett, B Wu, et al. Bbrv3: Algorithm bug fixes and public internet deployment. *Presentation in CCWG at IETF*, 117, 2023.

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

[13] 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.

[14] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. Rebuffering but not suffering: Exploring continuous-time quantitative qoe by user's exiting behaviors. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[15] 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.

[16] 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 18)*, pages 343–356, 2018.

[17] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. A unified congestion control framework for diverse application preferences and network conditions. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 282–296, 2021.

[18] Facebook. Mvfst. `https://github.com/facebook/mvfst`, 2024. [Online; accessed 31-Janurary-2024].

[19] 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.

[20] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. {ABC}: A simple explicit congestion controller for wireless networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 353–372, 2020.

[21] 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.

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

[23] IOVisor. Userspace ebpf vm. `https://github.com/iovisor/ubpf`, 2024. [Online; accessed 31-Janurary-2024].

[24] Jeffrey Jaffe. Flow control power is nondecentralizable. *IEEE Transactions on Communications*, 29(9):1301–1306, 1981.

[25] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[26] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[27] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *ICC'79; International Conference on Communications, Volume 3*, volume 3, pages 43–1, 1979.

[28] John Klippert. Advanced advanced calculus: Counting the discontinuities of a real-valued function with interval domain. *Mathematics Magazine*, 62(1):43–48, 1989.

[29] Stanley Ko. Mathematical analysis. 2006.

[30] Zhi Li, Christos Bampis, Julie Novak, Anne Aaron, Kyle Swanson, Anush Moorthy, and JD Cock. Vmaf: The journey continues. *Netflix Technology Blog*, 25(1), 2018.

[31] Linux. ebpf. `https://ebpf.io`, 2024. [Online; accessed 31-Janurary-2024].

[32] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. Multi-objective congestion control. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 218–235, 2022.

[33] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 229–244, 2021.

[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] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *USENIX winter*, volume 46, pages 259–270, 1993.

[36] Zili Meng, Yaning Guo, Yixin Shen, Jing Chen, Chao Zhou, Minhu Wang, Jia Zhang, Mingwei Xu, Chen Sun, and Hongxin Hu. Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning. *IEEE/ACM Transactions on Networking*, 29(2):723–736, 2021.

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

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

[39] MMSys. Bandwidth estimation for real-time communications. `https://2021.acmmmsys.org/rtc_challenge.php`, 2021. [Online; accessed 31-Janurary-2024].

[40] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. *SIGCOMM Comput. Commun. Rev.*, 45(4):311–324, August 2015.

[41] Christopher Müller and Christian Timmerer. A vlc media player plugin enabling dynamic adaptive streaming over http. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 723–726, 2011.

[42] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[43] OpenNetLab. Alphartc. `https://github.com/OpenNetLab/AlphaRTC`, 2024. [Online; accessed 31-Janurary-2024].

[44] Fernando Paganini, John Doyle, and Steven H Low. A control theoretical look at internet congestion control. In *Multidisciplinary Research in Control: The Mohammed Dahleh Symposium 2002*, pages 17–31. Springer, 2003.

[45] Feng Peng, Bingcong Lu, Li Song, Rong Xie, Yanmei Liu, and Ying Chen. Pacc: Perception aware congestion control for real-time communication. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*, pages 978–983. IEEE, 2023.

[46] Waleed Reda, Kirill Bogdanov, Alexandros Milolidakis, Hamid Ghasemirahni, Marco Chiesa, Gerald Q Maguire Jr, and Dejan Kostić. Path persistence in the cloud: A study of the effects of inter-region traffic engineering in a large cloud provider's network. *ACM SIGCOMM Computer Communication Review*, 50(2):11–23, 2020.

[47] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proc. IEEE INFOCOM*, 2016.

[48] Rayadurgam Srikant and Tamer Başar. *The mathematics of Internet congestion control*. Springer, 2004.

[49] Ao-Jan Su and Aleksandar Kuzmanovic. Thinning akamai. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, page 29–42, New York, NY, USA, 2008. Association for Computing Machinery.

[50] Ao Tang, Jiantao Wang, Steven H Low, and Mung Chiang. Equilibrium of heterogeneous congestion control: Existence and uniqueness. *IEEE/ACM Transactions on Networking*, 15(4):824–837, 2007.

[51] Tencent. Tquic. https://github.com/Tencent/tquic, 2024. [Online; accessed 31-Janurary-2024].

[52] Viet-Hoang Tran and Olivier Bonaventure. Beyond socket options: making the linux tcp stack truly extensible. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[53] Sara Vlahovic, Mirko Suznjevic, and Lea Skorin-Kapov. The impact of network latency on gaming qoe for an fps vr game. In *2019 Eleventh international conference on quality of multimedia experience (QoMEX)*, pages 1–3. IEEE, 2019.

[54] 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.

[55] Stephen J Wright. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34, 2015.

[56] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 710–723, New York, NY, USA, 2023. Association for Computing Machinery.

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

[58] Furong Yang, Zhenyu Li, Jianer Zhou, Xinyi Zhang, Qinghua Wu, Giovanni Pau, and Gaogang Xie. Disco: A framework for dynamic selection of multipath congestion control algorithms. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2023.

[59] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. Axiomatizing congestion control. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1–33, 2019.

[60] Jia Zhang, Yixuan Zhang, Enhuan Dong, Yan Zhang, Shaorui Ren, Zili Meng, Mingwei Xu, Xiaotian Li, Zongzhi Hou, Zhicheng Yang, et al. Bridging the gap between {QoE} and {QoS} in congestion control: A large-scale mobile web service perspective. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 553–569, 2023.

[61] Rui-Xiao Zhang, Changpeng Yang, Xiaochan Wang, Tianchi Huang, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. Aggcast: Practical cost-effective scheduling for large-scale cloud-edge crowdsourced live streaming. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 3026–3034, New York, NY, USA, 2022. Association for Computing Machinery.

[62] Tiesong Zhao, Qian Liu, and Chang Wen Chen. Qoe in video transmission: A user experience-driven strategy. *IEEE Communications Surveys & Tutorials*, 19(1):285–302, 2016.

[63] Yusheng Zheng, Tong Yu, Yiwei Yang, Yanpeng Hu, XiaoZheng Lai, and Andrew Quinn. bpftime: userspace ebpf runtime for uprobe, syscall and kernel-user interactions. *arXiv preprint arXiv:2311.07923*, 2023.

[64] Jianer Zhou, Xinyi Qiu, Zhenyu Li, Gareth Tyson, Qing Li, Jingpu Duan, and Yi Wang. Antelope: A framework for dynamic selection of congestion control algorithms. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

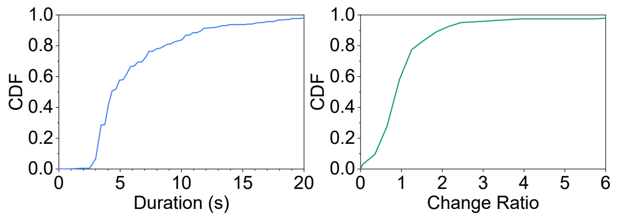
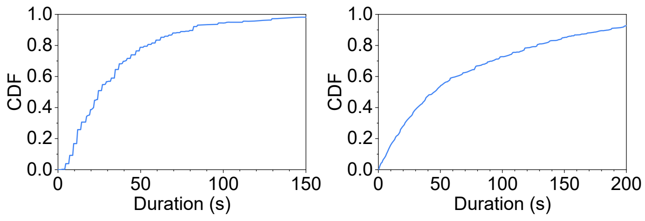


Figure 17: CDFs of segment duration and average throughput change ratio between adjacent segments, sliced by PELT [26] on production recorded traces



(a) Salsify Dataset [19] **(b)** Puffer Dataset [57]

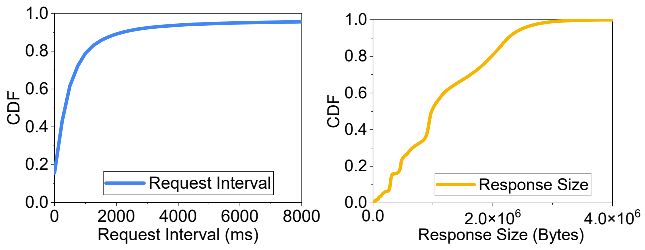
Figure 18: CDFs of segment duration from passive data analysis

A Production Network Characteristic

We analyze thousands of throughput traces sampled from production environments, segmenting the traces (after filtering out outliers) using the PELT method [26]. We display the duration of each segment and the ratio of change in average throughput between adjacent segments in Fig.17. It's apparent that the primary segment durations fall within a range of a few seconds to tens of seconds: over 90% of the segments last longer than 3 seconds, while more than 95% last less than 17 seconds. The changes between segments are also substantial: the ratio of change in average throughput exceeds 50% for over 80% of consecutive segments, and it's within 245% for 95% of successive segments. This data highlights that network conditions in production environments undergo frequent and significant changes. We have also conducted the same passive data analysis on publicly available datasets (Salsify [19] and Puffer [57]), and have obtained similar results, as shown in Fig.18.

B Workload Characteristics

We measure the traffic patterns of real services in our production environments, which illuminate the nature of request-response messaging traffic, e.g., file downloading, web browsing, etc. We present the characteristics of these real-world



(a) Distribution of the request interval **(b)** Distribution of the response size

Figure 19: Distributions of request and response characteristics

services, i.e., the frequency of requests sent and the size of the responses transferred between the client and the servers on the CDN in Fig.19. Fig.19a shows the CDF of the time interval between two consecutive requests sent by the client. The inter-sending time between requests reflects the density and diversity of requests initiated by the application. As we can see, 40% of the request inter-sending intervals are less than 200ms, and 36% of them are concurrent (0ms). 80% of all request intervals occur within one second, and any intervals longer than this can generally be attributed to user behaviors, such as different clicks [60]. Fig.19b shows the CDF of the response size. As we can see, around 30% of the responses are less than 100 KB and about 50% of the responses are larger than 1 MB. *Given the frequency of requests and the size of responses, applications continue to exhibit a high demand for bandwidth, characterized by an on-off pattern.*

C Case Study 1: Video conference on WebRTC

We carry out our first case study on a real-world video conferencing application from ACM MMSys'21 Grand Challenge [39] based on WebRTC. Specifically, we present the implementation of Mortise on GCC [13] and the experimental setup (§C.1). We also analyze the non-linear QoE model to evaluate the performance of the video conferencing (§C.2) and the evaluation results (§C.3).

C.1 Implementation & Setup

We deploy our framework on AlphaRTC [43], a fork of Google's WebRTC used by ACM MMSys. We implement the communication mechanism (i.e., the aggressiveness interpreter) on AlphaRTC with shared memory, while the remaining components of the framework are directly reused from the implementation in the TCP kernel stack.

We choose GCC as the underlying scheme of Mortise (i.e., Mortise-GCC). We map the aggressiveness to the step length (default is 1) in the additive increase state or the growth limit (default is 1.08) in the multiplicative increase state (since GCC will exclusively be in one of the two states when increasing). We also modify the decrease rate factor (default is 0.85) correspondingly. For brevity, we omit the details of the mapping between them.

We conduct our experiment with Mahimahi [42], using wireless traces from Orca [2] and DeepCC [3]. To ensure repeatability and comparability, we opt to transmit different videos via the conferencing application instead of capturing footage with a real camera. We set the base RTT as 40ms. We select 10 videos with different resolutions, each playing for 120 seconds, and we repeat the tests 20 times.

C.2 Non-linear QoE model

We employ a non-linear QoE function to evaluate the performance of the video conferencing application on WebRTC. The total QoE score S_{total} is derived from ACM MMSys [39] and PACC [45], which consists of video quality score S_{video}, frame delay score S_{delay} and frame drop score S_{drop}:

Application	QoE Metric	RELP with Throughput	RELP with Latency	RELP with Loss
RTC	SSIM [8, 19], VMAF [45], PSNR [8, 56]	logarithmic-like	-	non-linear
RTC	Stall Rate [38], Deadline Miss Rate [37]	-	piecewise	piecewise
RTC	Frame Delay [45]	inverse proportional	piecewise	piecewise
VoD	Rebuffering Ratio [14, 34]	inverse proportional	linear	non-linear
Web	RCT [60], PLT [42]	inverse proportional	linear	piecewise

Table 1: The relationship with throughput, latency, and packet loss rate of QoE metrics in typical application scenarios

$$S_{\text{video}} = vmaf \tag{10}$$

$$S_{\text{delay}} = \max(100 - \frac{d_{avg}}{3}, 0) \tag{11}$$

$$S_{\text{drop}} = 100 \times (1 - M) \tag{12}$$

$$S_{\text{total}} = \mu_1 \times S_{\text{video}} + \mu_2 \times S_{\text{delay}} + \mu_3 \times S_{\text{drop}} \tag{13}$$

where $\mu_1 = 0.5$, $\mu_2 = 0.3$ and $\mu_3 = 0.2$. $vmaf$ denotes the VMAF score [30] of the received video, rating from 0 to 100. M and d_{avg} denote the frame drop ratio and average frame delay (ms) of the received video, respectively.

Given that this QoE model incorporates various QoE metrics, it necessitates local mapping onto throughput and latency before extracting application preferences. Considering that frame drops in WebRTC are primarily influenced by factors like the encoder and the inability to adjust the bitrate in a timely manner, we disregard this when evaluating the application's preference between throughput and latency. For frame delay d_f, we can have the following:

$$d_f = \frac{\zeta}{T} + D + \theta \tag{14}$$

where ζ is the frame size and θ is the encoding delay, while T and D are throughput and latency, respectively. Since the transmission of frames is sequential, evaluating d_{avg} is nearly equivalent to evaluating d_f. For commercial hardware, the encoding delay is mostly stable. Moreover, given that the frame size is calculated based on throughput (i.e., bitrate) and frame rate (i.e., fps, frames per second), $\frac{\zeta}{T}$ can also be approximated as a constant. Thus, we can approximate the sum of θ and $\frac{\zeta}{T}$ as a constant c to have the following:

$$S_{\text{delay}} \approx \max\left(0, 100 - \frac{1}{3}(c + D)\right) \tag{15}$$

The score of VMAF tends to increase with a rise in bitrate (i.e., throughput). However, there isn't a universally accepted formula that delineates the relationship between the quality score and bitrate, as factors like video resolution and viewing context can significantly influence the perceived quality. Nevertheless, drawing from the VMAF-bitrate relationship model available in [30], we can derive the following empirical formula (the unit of T is $Mbps$):

$$S_{\text{video}} = vmaf \approx 41.06 + 30.53 \times \ln(T + 0.1481) \tag{16}$$

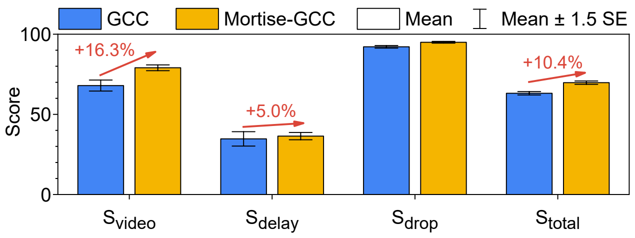


Figure 20: Detailed QoE Score of video conferencing

Hence, we can get the local mapping of QoE on throughput and latency:

$$\begin{aligned} S_{\text{local}} = &\mu_1(41.06 + 30.53 \times \ln(T + 0.1481)) + \\ &\mu_2\left(100 - \frac{1}{3}(c + D)\right) + \mu_3\left(100(1 - L)\right) \end{aligned} \tag{17}$$

Then the application latency-throughput preference hint λ at operating point (T_0, D_0, L_0) can be extracted directly:

$$\lambda = \left| \left(\frac{\partial S_{\text{local}}}{\partial D}\right)_{(T_0, D_0)} \Big/ \left(\frac{\partial S_{\text{local}}}{\partial T}\right)_{(T_0, D_0)} \right| \tag{18}$$

$$= \frac{\mu_2}{91.59\mu_1}(T_0 + 0.1481) = \frac{T_0 + 0.1481}{152.65} \tag{19}$$

Similarly, we can get the application loss-throughput preference hint β at the same operating point:

$$\beta = \frac{100\mu_3}{30.53\mu_1}(T_0 + 0.1481) = \frac{T_0 + 0.1481}{0.76325} \tag{20}$$

Intuitively, as throughput progressively increases, the corresponding gains in video quality, achieved by throughput enhancement at the cost of increased latency or loss, begin to diminish. Therefore, the application's preference gradually transitions from high throughput to low latency and less loss.

C.3 Evaluation Results

We present the average scores for Mortise-GCC and GCC on all 3 QoE metrics and the total QoE model in Fig.20. Mortise-GCC demonstrates a significant enhancement across all 3 key metrics: video quality score, frame delay score, and frame drop score, with respective increases of 16.3%, 5.0%, and 3.1%. These collective improvements result in a notable 10.4% increase in the overall QoE scores compared to GCC. Mortise-GCC also exhibits more concentrated scores across all its components, indicating a more stable performance.

The performance enhancement highlights the vital role of the application preference extractor and feedback control system in Mortise's design. When the bitrate is low, the extracted application preference tends towards throughput, prompting the feedback control system to make Mortise-GCC more aggressive. Mortise-GCC then attempts to increase its bitrate swiftly. A rapid rise in bitrate at lower bitrates can significantly enhance video quality, thereby drastically improving the VMAF score. While an aggressive bitrate growth may lead to increased frame delay, the relatively small frame size and minimal additional traffic make adjustments manageable. In other words, the potential loss from increased frame delay is less critical compared to the substantial potential improvement in video quality. Therefore, more aggressive sending can yield better application performance. As the bitrate escalates, the extracted preference increasingly leans towards latency. Consequently, the feedback control system gradually adjusts Mortise-GCC's decisions to be more conservative, favoring the maintenance of low frame delay. In this scenario, the video quality improvement from boosting the bitrate becomes relatively minor. Over-sending can significantly increase frame delay and even cause frame loss, drastically reducing application performance. In short, Mortise-GCC's capability to continuously adjust its aggressiveness in line with application preferences allows it to achieve comprehensive performance enhancement.

D Case Study 2: Web on QUIC

We conduct our second case study on a popular web application from [60] based on QUIC. Specifically, we present the implementation of Mortise and the experiment setup (§D.1). We analyze the piecewise QoE model used by the web application (§D.2) and the evaluation results (§D.3).

D.1 Implementation & Setup

We deploy the web application and Mortise on TQUIC, a production-ready QUIC implementation. Similarly, we only craft the aggressiveness interpreter with shared memory and reuse other components. We continue to use Copa as the underlying scheme of Mortise (i.e., Mortise-Copa) and employ the same mapping as in the TCP kernel stack. We compare our approach with different CCAs, including BBR [12], BBRv3 [11], Cubic [22] and Copa [7].

We conduct our experiment on Mahimahi [42] with traces from Orca [2]. Following the setup outlined in [60], we create 400 unique network conditions by combining loss rates ranging from 0-1%, RTTs varying between 10-300ms, and buffers within the range of 0.5×BDP to 2×BDP, to test the application performance under different CCAs. Each application is subjected to workloads generated based on the distribution provided in [60], each with a duration of 3 minutes. We repeat the experiment 5 times.

D.2 Piecewise QoE model

RCT is a commonly utilized metric for evaluating the QoE of web applications [60]. It measures the time interval between the initiation of a request and the complete receipt of the corresponding response. For a request and its corresponding response, RCT can be calculated as:

$$RCT = \frac{R}{T} + D \tag{21}$$

where R denotes the payload size of the response, while T and D correspond to throughput and latency, respectively. To achieve a minimal RCT, a larger response characterized by a higher R favors higher throughput, while smaller responses are more inclined towards lower latency.

Applications typically employ the average request completion time RCT_{avg} as their QoE model:

$$RCT_{avg} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{R_i}{T} + D\right) \tag{22}$$

where n represents the number of concurrent requests and R_i denotes the payload size of the i^{th} response.

It implies that the specific QoE model would correlate with the size distribution of concurrent requests at that moment. Consequently, the throughput-latency preference also fluctuates in accordance with the specific QoE model. Such variations in QoE models are primarily driven by user behavior (e.g., content accessed or access patterns).

The application preference hint λ at point (T_0, D_0) can be directly calculated as (since we don't have loss metrics in our QoE, β always equals 0):

$$\lambda = \left| \left(\frac{\partial RCT_{avg}}{\partial D}\right)_{(T_0,D_0)} \Big/ \left(\frac{\partial RCT_{avg}}{\partial T}\right)_{(T_0,D_0)} \right| = \frac{T_0^2}{\frac{1}{n}\sum_{i=1}^{n} R_i} \tag{23}$$

Evidently, among n responses, the application's preference tends to lean more towards high throughput when the proportion of large responses increases.

For QoE models subject to changes from non-network factors, collaboration from applications is crucial to leverage the real-time preference effectively. Applications should communicate the evolving QoE models to the application preference extractor during runtime.

D.3 Evaluation Results

We present the RCT distribution for all the CCAs in Fig.21. Mortise-Copa consistently adjusts to the most suitable operating point under various network conditions, demonstrating superior performance compared to all other algorithms. Mortise-Copa exhibits an improvement of 11.5% over Copa and a notable 25.3% over BBR. In these network scenarios, BBRv3 and Cubic underperformed, with Mortise-Copa showing substantial improvements of 75.2% and 71.0% over them, respectively. The effectiveness of Mortise primarily stems

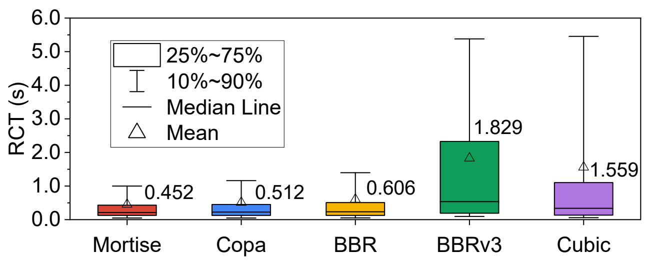


Figure 21: RCT of all responses in the Web application. The annotated value is average RCT.

from its real-time application preference extractor. In collaboration with the application, it continuously captures the ever-changing QoE model to compute real-time application preferences. The feedback control system then adjusts based on these preferences. This capability allows Mortise-Copa to balance RCT between large and small responses when they occur simultaneously. In such circumstances, it adopts a more conservative sending, ensuring that the bottleneck queue shared by large and small responses does not grow excessively long, thereby optimizing the RCT for small responses. In scenarios with only large responses, Mortise-Copa would aggressively utilize bandwidth to minimize RCT.