Pudica: Toward Near-Zero Queuing Delay in Congestion Control for Cloud Gaming

Shibo Wang[†‡], Shusen Yang[†], Xiao Kong[‡], Chenglei Wu[‡], Longwei Jiang[‡], Chenren Xu[*], Cong Zhao[†], Xuesong Yang[•], Jianjun Xiao[‡], Xin Liu[‡], Changxi Zheng[∘], Jing Wang[‡], Honghao Liu[‡]
[†]*Xi'an Jiaotong University*, [‡]*Tencent Inc.*, [*]*Peking University*, [•]*Bonree*,
[∘]*Pixel Lab, Tencent America, and Columbia University*

Abstract

Congestion control (CC) plays a pivotal role in cloud gaming services. However, existing CC methods often cause self-induced bottleneck queuing. As a result, they may largely delay game frame transmission and undermine the player's gaming experience. We present a new end-to-end CC algorithm named *Pudica* that strives to achieve near-zero queuing delay and high link utilization while respecting cross-flow fairness. Pudica introduces several judicious approaches to utilize the paced frame to probe the bandwidth utilization ratio (BUR) instead of bandwidth itself. By leveraging BUR estimations, Pudica designs a holistic bitrate adjustment policy to balance low queuing, efficiency, and fairness. We conducted thorough and comprehensive evaluations in real networks. In comparison to baseline methods, Pudica reduces the average and tailed frame delay by 3.1$\times$ and 4.9$\times$ respectively, and cuts down the stall rate by 10.3$\times$. Meanwhile, it increases the frame bitrate by 12.1%. Pudica has been deployed in a large-scale cloud gaming platform, serving millions of players.

1 Introduction

The digital world has been largely on the "cloud", a metaphor suggesting that storage- and computation-heavy applications should be in the ether, available through the Internet whenever and wherever we need them. The recent emergence of next generation internet infrastructures such as WiFi-7, 5G, and full fibre broadband catalyzes more cloud-based applications, and one of the most notable is *cloud gaming*. Thanks to the academic research effort [1–5] and industrial investment [6–9], cloud gaming has already gained worldwide popularity while still under rapid growth [10].

In the cloud gaming setup (see Fig. 1), the end device collects user operations (e.g., mouse clicks and finger taps) and immediately sends them to the cloud server, which runs the entire game engine. Then, the corresponding game frame is rendered on the server, streamed to the end device, and displayed locally. In this way, the often heavy-duty game engine is moved to the cloud; the end device is only responsible for user interactions, and thus, it can be lightweight, portable, and low-cost while still providing engaging gaming experiences.

This is a promising paradigm for gaming, but a key premise of this paradigm is that the game frames rendered on the cloud must be transmitted to the end device at a *consistent ultra-low* delay, typically within 50 *ms*. High frame delays, even if occasional, would severely undermine the gaming interactivity and in turn the player's quality of experience (QoE) (see our studies in §2.3). While many congestion control (CC) algorithms [11–17] have been proposed to restrict the inflation of bottleneck queuing, none of these solutions actually offers ultra-low frame delays in both average and tail (§5.2).

Prior delay-aware CC methods [11, 12, 18, 19] strive for approaching an *ideal* state with high link utilization, zero queuing, and fair allocation, which is indeed a state we long for as well. However, these methods often require a substantial queue buildup *initially* to probe the link status before making adjustments toward the ideal state. The periodic nature of overshooting-based probing actions leads to frequent *self-induced* queuing. Recently, some CC methods [13,20] achieve low latency in single-flow or isolated environments, but they fail to maintain low queuing when multiple flows coexist.

To achieve both stringent delay control and the highest possible bitrate, a cloud gaming system must be equipped with a *carefully* designed CC algorithm that meets the following three requirements. Firstly, it should be able to reach high link utilization (i.e., efficiency) without resorting to overshoot-based network probing. Secondly, heavy queuing should be also prevented when multiple homogeneous flows run concurrently. Thirdly, when the link condition suddenly degrades, the CC agent must promptly react to it and drain the bottleneck queue (if any) as soon as possible. The last is crucial because there exists a large amount of urgent bandwidth degradation on the Internet (see our measurements in §2.2).

With the above principles in mind, we design a new CC algorithm, which we call *Pudica*[1], for large-scale cloud gaming services. Under the constraint of consistent low *frame* delay, Pudica achieves high bandwidth utilization and cross-flow fairness (if applicable) in both single-flow cases and scenarios where it contends with other homogeneous flows. Pudica has been successfully deployed on a commercial cloud gaming platform (Tencent START [9]), serving millions of players.

Unlike existing CC frameworks [11, 12, 18], which intentionally trigger queue buildups to probe the link condition, we estimate the link's bandwidth utilization ratio (BUR) rather than the bandwidth *per se*. This signal has been exploited in [22,23] with explicit notifications, but in the context of end-to-end CC design, we propose a new probing method with

[1]Pudica is a plant known for its sensitivity and rapid movement [21].

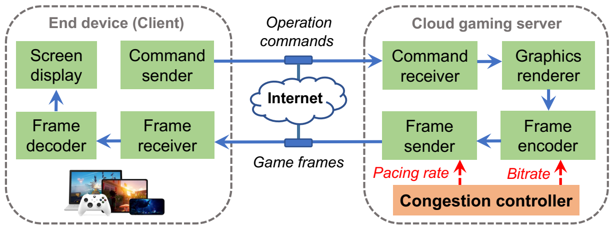


Figure 1: A typical cloud gaming system. The congestion controller adjusts both frame bitrates and packet pacing rates before transmitting game frames from the gaming server to the end device.

adaptive pacing control and complementary probe packets to estimate the BUR based on end-side feedback.

Based on the BUR estimation, Pudica multiplicatively increases the bitrate to exponentially ramp up the link utilization *while* ensuring that it remains below 100%. Then, Pudica can sustain the state without heavy queuing, and at the same time, quickly converge toward cross-flow fairness. This is achieved by applying additive increase (AI) and multiplicative decrease (MD) *simultaneously* and *adaptively*. In addition to smoothed BUR estimations, *short-term* BUR signals are also utilized in Pudica to make prompt reactions to any overshoots or urgent link degradation. The proposed mechanisms, including temporary bitrate fallback and active queue draining, facilitate the minimization of queuing delay.

We implemented Pudica and several baselines, including Copa [11], Salsify[2] [13], and SQP [20], on Tencent START cloud gaming production [9] for A/B tests. The final evaluation involved real-world wired and wireless networks, and more than 57,000 gaming sessions. In comparison to the baselines, our experiments confirm Pudica's advantages: **i)** It reduces the average frame delay, the 95%-tailed frame delay, and the 99%-tailed frame delay by 3.1×, 5.1×, and 4.7×, respectively. **ii)** It reduces the percentages of frames whose transmission to the end device exceeds 100 *ms* and 200 *ms* by 6.2× and 14.4×, respectively. And **iii)** it increases the frame bitrate by 12.1%, better utilizing networks' capacity.

Furthermore, we used a large-scale, in-the-wild network testing platform [24] to assess Pudica's efficiency and fairness. It shows that Pudica offers a superior balance between link utilization, cross-flow fairness, and convergence speed. Additionally, we conducted comprehensive microbenchmarks to justify the gains offered by Pudica.

2 Background and Related Work

In this section, we introduce the control variables, network characteristics, and control objectives of cloud gaming systems, which necessitates a new CC algorithm. We also review related CC algorithms and discuss their limitations.

2.1 Congestion Control for Cloud Gaming

Unlike the traditional transport-layer CC, the cloud gaming CC agent operates on the application layer, and controls two factors on the fly, namely the *frame bitrate* and *packet sending pace* (see Fig. 1). The bitrate is a parameter to the image compression process that encodes the stream of game frames before sending them to the end device. A lower bitrate reduces the bandwidth consumption but also sacrifices displayed quality. The game frames after compression are chopped into network packets, and the sending pace controls how hastily the packets are sent to the network. Typically, the sending interval of packets within a frame is the same. Thus, we measure the pace based on the sending behavior of the last packet for each frame. We quantify the pace using a *pace multiplier* ρ, a scalar indicates that the last packet is sent to the network within the time duration of L/ρ, where L is the interval of frame sending (16.67 *ms* for the frame rate of 60). For example, a pace with a multiplier of two means that all packets of a frame are sent over within 8.34 *ms* after encoding. *Given* a pace multiplier, the exact packet sending interval (a.k.a., packet pacing rate) depends on the number of packets within a frame, which is in turn up to the frame bitrate, or more precisely, the frame size.

2.2 Network Measurement in Real Gaming Systems

We present two observations from real cloud gaming systems, which greatly influence the rationale behind our CC design.

Short base RTTs. Edge computing has become the de facto infrastructure for cloud gaming services in industry [2, 25], which dramatically shortens the feedback loop of CC agents. We conducted measurements of the base RTT (a.k.a, minimal RTT) for over one million real cloud gaming sessions. As depicted in Fig. 3, the base RTT for 50% and 90% gaming sessions is below 10 *ms* and 20 *ms*, respectively, for both Ethernet and WiFi networks. Such *unique* network characteristics play an important role in our CC design. It not only helps us narrow down the scope of issues to be addressed but also opens up new possibilities for achieving a more suitable CC solution for cloud gaming systems.

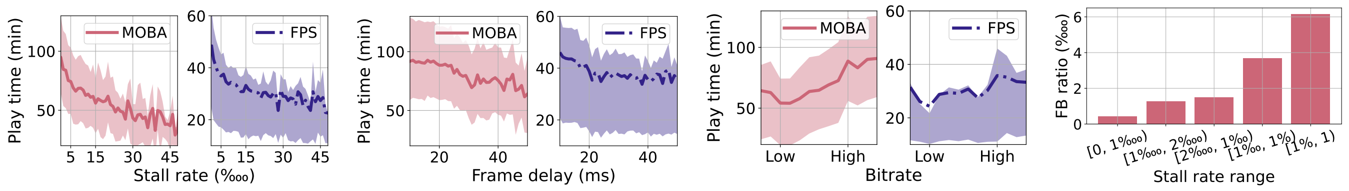
Frequent urgent bandwidth decreases. The available bandwidth frequently encounters urgent, large reductions over the Internet, e.g., due to channel degradation. We carried out the consistent bandwidth measurement by mildly flooding-based probing methods. The bandwidth was estimated and recorded with a granularity of 100 *ms* interval. As shown in Fig. 4, approximately 5.6% of Ethernet users and 35.5% of WiFi users encounter at least five occurrences of dramatic bandwidth reduction (i.e. reduction by more than 50% within 100 *ms*) per minute, which poses a non-trivial challenge to achieving consistent low latency in real-world networks.

2.3 Control Goals through Real-World Case Studies

In this section, we crystalize our congestion control goals through real-world case studies of user engagement on Tencent START cloud gaming app [9].

We conducted case studies on two popular games, a multiplayer online battle arena (MOBA) game and a first-person

[2]In this paper, *Salsify* refers to its frame size control part solely.



(a) Stall rate v.s. Playing time (b) Avg. frame delay v.s. Playing time (c) Avg. bitrate v.s. Playing time (d) Stall rate v.s. Feedback ratio

Figure 2: A case study on cloud gaming QoE analysis: the effects of stall rate (i.e., the frame ratio with delay >100ms), average frame delay, and average bitrate on playing time (as a proxy of QoE). The dataset involves two hot games (i.e., MOBA and FPS) and one million gaming sessions from a production cloud gaming service. Transmission-unrelated factors, such as decoding delay, are excluded from these metrics.

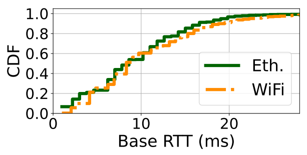


Figure 3: Base RTTs in edge-based cloud gaming services.

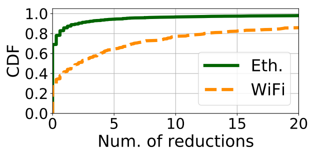


Figure 4: Num. of bandwidth reductions (≥ 50%) per minute.

shooting (FPS) game. We collected logs of more than a million gaming sessions to study how network performance affects user engagement in gaming. Following [26], we use the user playing time (i.e., the length of a gaming session) to measure user engagement. We consider three metrics for network performance: the average delay of game frames, the average bitrate, and the stall rate. The last measures how often a player experiences a *stall*, defined as the percentage of frames whose delivery to the user device is delayed by more than 100 *ms*.

The results of our studies are plotted in Fig. 2. The stall rate has a significant impact on user engagement (Fig. 2a): as the stall rate increases linearly, the playing time rapidly decreases in a superlinear manner. Average frame delay also affects gaming engagement (Fig. 2b). But when the average delay is sufficiently small (e.g., <40 *ms*), the negative impact caused by an increased average delay is not as significant as the stall rate, which reflects the long-tailed portion on the distribution of frame delays (as opposed to the average delay). Not surprisingly, the bitrate is also crucial to the user engagement, since it determines the image display quality (Fig. 2c). Lastly, Fig. 2d illustrates the percentage of sessions that received user complaints with respect to different ranges of stall rates. When the stall rate is in $[1‰, 1\%)$, the percentage of user-indicated unappealing gaming sessions is 10× more than the percentage when the stall rate is within $[0, 1‱)$.

Our CC goals. To reduce the frame delay and stall rate, we aim to achieve ultra-low (or *nearly zero*) queuing delay at the bottleneck. To achieve the highest possible bitrate, we seek to maximize bandwidth utilization; however, it should be pursued only *after* the achievement of low frame delays. A high bitrate with a high frame delay or stall rate is meaningless for cloud gaming. We also value fairness among homogeneous flows. It is noteworthy that, we specifically focus on achieving *frame-level* zero queuing, which directly impacts the QoE. This means that the queue induced by a frame should be drained by the time the next frame arrives at the bottleneck, rather than aiming for no queuing for every packet, although these two levels of queuing are correlated.

2.4 Existing CC Solutions and Their Limitations

Many CC algorithms have been proposed with the aim of reducing end-to-end delay, but they cannot simultaneously fulfill the three specific requirements we mentioned in §1.

Delay-bounding CC. In previous studies, various *delay-bounding* CC algorithms [11, 18, 19, 27–34] has been proposed. Most of these algorithms, such as BBR [18] and PCC Vivace [19], *periodically* increase packet sending rate beyond the estimated bandwidth in order to dynamically probe network bandwidth. This strategy, however, leads to frequent queue buildup, and thus may cause the delay to oscillate persistently even when the bandwidth stays stable. Other algorithms (e.g., Vegas [27], Fast [28], and Copa [11]) use packet delay as the congestion signal to avoid bufferbloat. When the bottleneck queue is nearly empty, these methods tend to *recklessly* raise the sending rate (even with an increasing step size) until the queue is constructed, since these delay-based methods can only accurately assess link utilization when there is sufficient queuing in the network. Thus, their CC frameworks are fundamentally unable to maintain zero queuing.

RTC-oriented CC. Recent years have also witnessed the development of CC algorithms specifically for real-time communication (RTC) applications [12–17,35]. GCC [12] reduces the bitrate only when the delay variation, rather than the delay *per se*, becomes high. So, GCC is unable to empty the queue promptly when an urgent congestion occurs. Similar to packet delay-based methods, the network estimator in GCC works only when there exists enough queuing in the network. As a consequence, GCC has to often introduce queue buildups by itself (as illustrated in Fig. 16, Appendix).

Salsify [13] (built upon Sprout [29]) adjusts frame sizes based on the estimation of packet inter-arrival time. Using samples in a short time scale for estimation, Salsify is sensible to network changes and can respond to congestion quickly. However, in real networks, the inter-arrival time at the packet level suffers from significant fluctuations, which raise the chances of estimation mistakes. Additionally, Salsify, which is designed based on the assumption of independent queues, is unable to detect other competing flows unless they intersect. Furthermore, the aggressiveness of Salsify magnifies the impact of estimation errors stemming from the aforementioned issues, ultimately leading to suboptimal performance.

SQP [20] utilizes frame-based packet trains to estimate available bandwidth and adjusts the frame bitrate with AIMD-style updates. This approach may allow SQP to achieve high link utilization and avoid heavy queuing in an isolated environment. However, when multiple flows coexist, SQP faces challenges in accurately estimating the available bandwidth, especially when the flows are staggered. This may result in overshoots and queue buildups. Furthermore, SQP tries to achieve competitive bandwidth shares when competing with queue-building flows and may incorrectly identify a network degradation as the competition of buffer-filling flows (as explained in Fig. 12). Consequently, SQP fails to promptly drain the bottleneck queue in such situations.

3 Challenges and Rationale

Before diving into Pudica's algorithmic details (in §4), we describe the challenges that Pudica aims to address for clouding gaming applications and the rationale that leads to its design.

3.1 Network Probing and Estimation

Existing bandwidth estimation methods are unable to sustain a nearly empty bottleneck queue at the frame level. In Pudica, we opt to estimate the BUR (i.e., bandwidth utilization ratio) rather than the bandwidth itself, and to achieve this, we design a new network probing method.

Limitations of traditional methods. Many CC algorithms aim to estimate the available bandwidth or capacity between two ends of a connection [18–20]. Here, to eliminate ambiguity, we refer to [36] to define the term "available bandwidth" as the maximum rate that the path can provide to a flow, without reducing the rate of the rest of the traffic. Similarly, "available capacity" is defined as the amount of data that can be inserted into a network path at a certain time, so that the transit delay of these packets remains within a specified maximum permissible delay (for further details, refer to [36]).

For estimating bandwidth or capacity, most of the existing frameworks require the intentional and periodic introduction of heavy queuing at the bottleneck to assess whether the link is fully utilized. This network probing method is not at our disposal, because we aim for a *consistently* empty queue. Recently, a few low-latency CC methods try to probe the link condition without resorting to heavy queuing [13, 20]. However, these designs often begin with a potential assumption that the sender runs in a single-flow environment, without giving due consideration to the collective dynamics of multiple flows. This oversight may result in an *overestimation* of available network resources when multiple streams coexist.

In fact, even with exact knowledge of the remaining bandwidth or capacity, we cannot guarantee a rate increase without surpassing the available resources. This is due to the fact that, as an end-to-end CC agent, we lack information regarding the number of flows utilizing the identical bottleneck link,

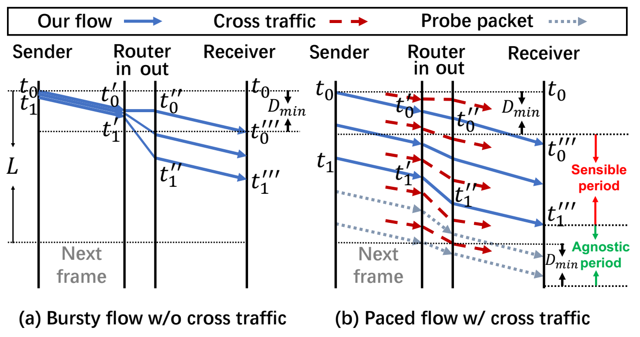


Figure 5: An illustration of our BUR probing and estimation method with the router acting as the bottleneck.

and consequently, an appropriate allocation for aggregated resources remains unattainable. This limitation is inherent to the signal of available bandwidth/capacity.

What to estimate? As will be mentioned repeatedly, it is the requirement of nearly zero queuing delays for game frames that profoundly shapes Pudica's design in many aspects. As the prerequisite for control actions, not only should the *probing process* keep no queuing, but the chosen signal after probing must also enable the rate controller to avoid queue buildups during the *efficiency-convergence process*, particularly when multiple streams coexist.

Inspired by VCP [23], Pudica probes the BUR as a feedback signal for CC, rather than the bandwidth *per se*. The BUR, a scalar, is defined as the ratio of current bandwidth usage to the link capacity, which provides an indicator of the precise level of link utilization. It not only indicates whether congestion is occurring (congested when BUR >1), but also distinguishes between different utilization regions. As proved by [23], by leveraging the BUR information, we can achieve efficiency and fairness under the constraint of consistently low queuing, regardless of the number of concurrent flows or link capacity.

How to probe? Our goal is to estimate BUR by relying solely on end-to-end feedback. Basically, a queuing delay greater than zero serves as a reliable indicator of the full-load duration at the bottleneck. Since we aim for nearly zero queuing at the frame level, for probing BUR, packet-level queuing delay must be introduced judiciously.

To illustrate this, we present an example in Fig. 5, where the router acts as the bottleneck. In the absence of cross traffic (refer to Fig. 5a), all packets of a frame are transmitted in a bursty manner by the sender between time t_0 and t_1, and they reach the router between time t'_0 and t'_1. Due to the limitation of network bandwidth, the queuing time of the last packet can be denoted as $(t''_1 - t''_0)$, during which the link utilization is 100%. In this case, the bandwidth utilization can be calculated as $(t''_1 - t''_0)/L = (t'''_1 - t'''_0)/L = (t'''_1 - t_0 - D_{min})/L$, where D_{min} is the minimum one-way delay from the sender to the receiver, and L is the interval of frame sending.

However, when cross traffic exists, traffic arriving at the router after t'_1 cannot be detected by the receiver. To address this issue, we intentionally slow down the frame transmission slightly, allowing a portion of the cross traffic to arrive between t'_0 and t'_1, as depicted in Fig. 5b. This enables the receiver to observe the presence of such traffic. Although extending the time period between t_0 and t_1 allows for more cross traffic to arrive in this period, once the instantaneous utilization drops below 100%, the packet-level queuing delay becomes zero, and then we lose the ability to determine the level of bandwidth utilization accurately. Therefore, our goal is to send each frame as slowly as possible while maintaining a packet-level queuing delay greater than zero (at least for the last packet of each frame). To achieve this, we dynamically and adaptively adjust the pace multiplier. As a result, we can still *basically* estimate the BUR as $(t'''_1 - t_0 - D_{min})/L$, which remains consistent with the no-cross-traffic cases.

Nonetheless, we are actually uncertain about the cross traffic arriving after t'_1. To complement this, we employ a small number of extra probe packets beyond frame data to measure the queuing time and robustify the BUR estimation. Even though a competing flow that consistently sends at a low rate and causes zero queuing delay may still go undetected, this complementary probing mechanism can greatly reduce the occurrence of BUR underestimation (see Fig. 7). Furthermore, the lack of awareness of such low-rate flows diminishes as the link utilization approaches 100%.

3.2 Bitrate Adaptation based on BUR Estimations

Basically, Pudica utilizes BUR estimation to dynamically adjust the frame bitrate. When BUR is low, we multiplicatively increase the rate for efficiency; when BUR is high (but less than one), we operate AI and MD for fairness; when BUR exceeds one, we reduce the bitrate to quickly drain the bottleneck queue. Within this framework, Pudica introduces several meticulously designed methods to minimize queuing delay.

Coping with queuing caused by possible estimation errors. Despite significant improvements in the accuracy of BUR estimation, we acknowledge that there is still a gap between the predicted value and the actual value. Therefore, we recognize the need for adaptive steps in bitrate upgrading, i.e., the step should be smaller (applied more cautiously) as the BUR increases. Furthermore, we design a temporary *fallback mechanism* to minimize the adverse effects of potential estimation errors. Specifically, Pudica timely and moderately reduces the bitrate whenever it detects a significant delay in any individual frame. To enhance resilience against jitters, this reduction is limited to the subsequent frame only, ensuring its transient nature. This fallback strategy is particularly useful for cloud gaming due to the smaller RTT (§2.2). With shorter feedback loops, we can effectively manage the effects of minor overshoots through prompt corrections.

Balancing fairness and queuing. A common way to promote fairness in CC design is through additive increase and multiplicative decrease (referred to as *AI/MD*) of the bitrate. It additively increases the bitrate until *a high queuing delay* (or overflow-induced packet loss) is detected, and then it switches to multiplicatively decrease the bitrate.

Again, for cloud gaming this strategy falls short, because we wish to minimize queue buildups. We therefore propose a different strategy: While we additively increase the bitrate, we do not wait until the queue is built up to begin the MD operation. Instead, we apply MD earlier, concurrently with each AI action, to proactively free up a portion of bandwidth before the bitrate causes any queue buildup. In this way, MD-based de-allocation occurs simultaneously with AI-based allocation, executed more frequently as well. Thereby, fairness across competing flows can be reached in a faster way while avoiding excessive queuing. To distinguish it from the traditional AI/MD, this strategy is referred to as *AI-MD*.

Yet, a naïve implementation of AI-MD with fixed MD amplitudes and AI steps may hinder link utilization. For instance, in high-bandwidth scenarios, the de-allocation amount from the MD operation may exceed AI's allocation, leading to a bitrate decrease even with a low utilization of overall bandwidth. Our remedy to this limitation is by applying bitrate AI in an adaptive manner. The step of AI gradually increases from an initial small value until the link is close to being fully utilized. This approach ensures high utilization and low queuing during the convergence process toward fairness.

Handling bandwidth fluctuations. As a commercially deployed system, Pudica must be able to cope with frequent network fluctuations, especially urgent bandwidth degradation (which has been shown in §2.2). When the link degrades, a belated bitrate reduction would result in severe queuing; conversely, a timely response to this degradation would significantly reduce the tailed delay and stall rate.

To achieve a timely response to bandwidth reduction, Pudica relies on not only the smoothed BUR estimation but also the BUR feedback from the latest frames, to judge if congestion has occurred. When the BURs of three consecutive frames both exceed one, Pudica will reduce the bitrate to a value below the packet receiving rate. The underlying insight is that when the link utilization exceeds 100%, the short-term receiving rate can serve as a dependable indicator of the achievable maximum throughput. At this time, if every flow in the network can set its bitrate below the receiving rate, the queue at the bottleneck can be drained quickly.

4 Pudica Design

We propose a new probing framework that achieves high-accuracy BUR estimation. By leveraging both smoothed (from long-term history) and short-term BUR estimations, we design multiple bitrate control strategies to enhance efficiency and fairness while minimizing the queuing delay. We provide a brief overview of our control policy:

- MI for efficiency when smoothed BUR (or $\widetilde{R}$) is low;
- AI-MD for fairness when $\widetilde{R}$ is high but not more than one;

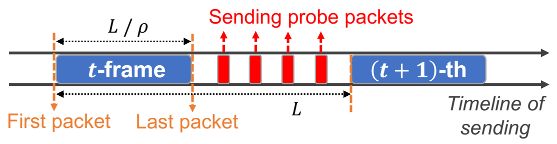


Figure 6: Pudica sends several payload-free probe packets beyond frame data during the intermission of frame sending to detect the existing competing flows during the agnostic period.

- MD for queue draining when short-term BUR exceeds one.

4.1 BUR Probing and Estimation

Pudica utilizes all packets within each game frame as a short burst (namely a packet train) to probe the link utilization (i.e. BUR). To improve the accuracy of BUR estimation, Pudica introduces an innovative adaptive pacing control algorithm, accompanied by a complementary approach that uses a small number of probe packets to detect competing flows.

As illustrated in Fig. 5 and §3.1, during the period of $[t_0 + D_{\min}, t_1''']$, the queuing delay keeps non-zero, and in turn, the bottleneck link is fully utilized. We refer to this period as the *sensible period*. On the contrary, $[t_1''', t_0 + L + D_{\min}]$ is the *agnostic period*, during which there is no feedback from the network. We adaptively adjust the pace multiplier such that it introduces just a slight amount of packet-level queuing for effective probing while maximizing the duration of the sensible period. In the agnostic period, we insert several additional probe packets without payload to measure the potential queuing time during this period.

BUR estimation. We estimate the BUR of a frame (denoted by R) by calculating the difference between the transmission delay of this frame and the physically minimal delay:

$$R = \frac{D - D_{min}}{L}. \tag{1}$$

Here, D is the one-way frame delay (in *sec*), i.e., the duration from the sent time of the first packet to the received time of the last packet for this frame. D_{min} (in *sec*) is estimated as the smallest packet one-way delay over the period of ten seconds. L is the interval of frame sending (in *sec*). There is no clock synchronization issue here, as the same errors for both D and D_{min} cancel each other out.

Adaptive pacing control. To extend the sensible period while sustaining a slight burstiness, Pudica dynamically sets the pacing multiplier ρ as:

$$\rho = \frac{\gamma_\rho}{\min(\, R,\ 1\,)}, \tag{2}$$

where γ_p is a constant greater than one, empirically assigned a value of 1.25. Using this method, the sending duration between the first packet and the last packet in a frame (i.e., $t_1 - t_0$ in Fig. 5) is slightly shorter than the queuing delay (i.e., $D - D_{min}$), which introduces just a slight packet-level queuing in the bottleneck buffer. By bounding the denominator up to one, all packets of each frame are sent within the frame interval, to avoid superfluous waiting time at the sender.

Competing flow detection with probe packets. To detect potential competing flows after the sensible period, Pudica sends N_{packet} (set as four) payload-free probe packets during the agnostic period, as illustrated in Fig. 6. These probe packets are sent out evenly, i.e., the sending interval T_{packet} (in *sec*) between probe packets is set to be $\frac{(1-1/\rho)\times L}{N_{packet}+1}$.

Then, Pudica uses the feedback of probe packets to robustify the BUR estimation within this frame interval. Let D_i denote the one-way delay (in *sec*) of the i-th probe packet ($i = 1, \ldots, N_{packet}$). If the competing flows that occupy the network resource during the agnostic period block the probe packets, the queuing delay T_i, i.e., $D_i - D_{min}$, for the i-th probe packet will be non-zero and should be taken into account.

However, the queuing delay also occurs when probe packets are blocked by the game frame data sent by Pudica itself. Such queuing delay needs to be removed as it has already been counted in the sensible period. If the frame data delays the i-th probe packet, the actual queuing delay caused by competing flows is the duration (in *sec*) between the arrival of the frame's last packet to the arrival of the i-th probe packet. Such delay is smaller than $D_i - D_{min}$ in this case, defined as H_i. So far, a reasonable queuing delay can be expressed as:

$$T_i = \min(\, D_i - D_{min},\ H_i). \tag{3}$$

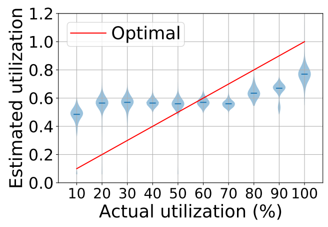
Furthermore, if the queuing delay of the i-th probe packet (i.e., T_i) is larger than the packet sending interval (i.e., T_{packet}), it will also affect the queuing delay of the following probe packet (i.e., T_{i+1}). To avoid repetitively counting the queuing delay that has been already experienced by the previous probe packets, T_i should be bounded by T_{packet} in computation. The rectified queuing delay is finally defined as:

$$T_i = \min(\, D_i - D_{min},\ H_i,\ T_{packet}\,). \tag{4}$$

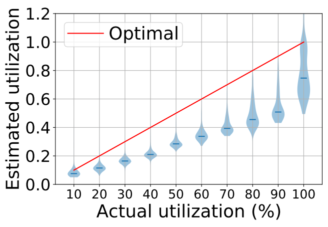
With the above analysis, Pudica can leverage the cross-traffic-induced queuing delay of probe packets to obtain a more accurate BUR estimation, i.e., $R^{corrected}$:

$$R^{corrected} = R + \Sigma_{i=1}^{N_{packet}} \frac{T_i}{L}. \tag{5}$$

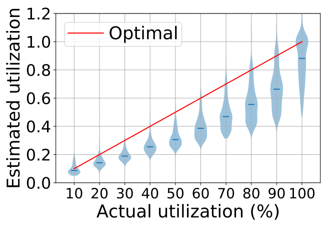
Validation of usefulness. We validated the effectiveness of our BUR estimation method through a large-scale Internet test. The details of the validation methodology are explained in Appendix A. Fig. 7 depicts the BUR estimation performance of different probing methods. Our probing approach (see Fig. 7d), which incorporates both adaptive paces and extra probe packets, outperforms other alternatives in accurately estimating BUR. Furthermore, as the actual BUR increases, the estimation distributions exhibit reduced variance, indicating enhanced reliability. Fixed 2× pacing (used in SQP [20], see Fig. 7a) and burst sending (see Fig. 7b) are prone to unawareness of competing flows and consequently lead to BUR underestimation, particularly in high-utilization scenarios. The use of probe packets not only corrects unawareness-induced BUR underestimation, but also guides the pacer toward smaller multipliers. This, in turn, extends the sensible period and enhances the precision of BUR estimation.



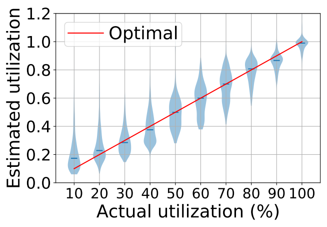
(a) Fixed paces (2× pacing).



(b) Fixed paces (burst sending).



(c) Adaptive paces (w/o probe packet).



(d) Pudica (complete).

Figure 7: Validation of various BUR estimation methods. Each method involves more than 2,000 groups of test data in real networks. Our method, which combines adaptive paces and probe packets, significantly improves the accuracy of BUR estimation over the alternatives.

4.2 Bitrate Adaptation over the Smoothed BUR

In this section, we introduce how Pudica leverages a *smoothed BUR estimation* as an indicator to dynamically adjust its bitrate for optimizing link utilization and cross-flow fairness while sustaining near-zero bottleneck queuing.

Smoothed BUR computation. To obtain the smoothed BUR (a scalar, denoted by $\widetilde{R}$), we utilize historical BUR estimation samples from N_{sample} frames over the past T_{wd} *ms*. T_{wd} is set to 200, as most flows have base RTTs less than 200 *ms*. This choice strikes a balance between the likelihood of awareness for competing flows and the freshness of the data [23]. We employ the weighted average method to calculate $\widetilde{R}$ as:

$$\widetilde{R} = \sum_{k=1}^{N_{sample}} \omega_k \times R_k \times \frac{B}{B_k}, \tag{6}$$

with weight coefficients ω_k subject to $\sum_{k=1}^{N_{sample}} \omega_k = 1$ (detailed in Appendix B). Here, R_k and B_k represent the BUR feedback and bitrate of the k-th frame, respectively. B is the current bitrate for frame encoding, which is dynamically updated on the fly. To address possible differences between B and B_k (both in units of *Mbps*), we use the ratio of the two values to rectify any deviation and ensure that the smoothed BUR estimation aligns with the current state.

Multiplicative increase (MI) for efficiency. Similar to VCP [23], Pudica decouples efficiency control and fairness control. When the smoothed BUR estimation (i.e., $\widetilde{R}$) is low, Pudica prioritizes efficiency over fairness, by adjusting bitrate with MI operations. Then, when $\widetilde{R}$ is high but less than one, the goal of Pudica is to converge toward fairness while maintaining nearly full utilization (will be introduced later).

Pudica employs a threshold, denoted by α, to determine whether to activate the efficiency control or fairness control phase. The value of α is empirically determined to be 0.85. When $\widetilde{R} \le \alpha$, the bitrate is multiplied to rapidly converge toward efficiency, which can be expressed as:

$$B^{new} = B \times (1 + \xi), \tag{7}$$

$$\xi = \gamma_{MI} \times \frac{(\alpha + 1)/2 - \widetilde{R}}{\widetilde{R}}, \tag{8}$$

where γ_{MI} is a discounting coefficient, empirically set as 0.3. Note that the next adjustment is postponed until the feedback regarding the current adjustment is received, to prevent over-aggressive rate increases and mitigate bitrate oscillation.

Simultaneous AI and MD for fairness. When $\widetilde{R} > \alpha$, we do not apply the traditional AI technique which linearly increases the bitrate until the queue is built up. Instead, Pudica operates AI and MD bitrate adjustment *simultaneously* (namely AI-MD), by replacing the *fixed* AI step (used in classical AI/MD frameworks) with an *adaptive* step $\mathcal{A}$:

$$B^{new} = B + \mathcal{A}, \tag{9}$$

$$\mathcal{A} = I - \gamma_{MD} \times B, \tag{10}$$

where I is the linear part of the adjustment step (in *Mbps*), and γ_{MD} is the MD parameter of the multiplicative part (set to be 0.05, empirically). Simultaneously performing AI and MD in one step significantly augments the prospects for bandwidth reallocation, thereby accelerating the convergence to fairness. In practice, we enforce both upper and lower bounds on $\mathcal{A}$ to prevent excessive bitrate oscillation during this stage.

Given an MD ratio (i.e., γ_{MD}), a fixed linear increase (i.e., I) may be too small to fully utilize bandwidth for a high link rate, or conversely, it may cause rate oscillation and delay spikes for low link bandwidth. To solve this dilemma, we design an AI step adaptation mechanism to match the link rate. Specifically, the value of I is determined by the equation $(B_{max} + \frac{2^\tau}{log(B)}) \times (\frac{\gamma_{MD}}{2})$. Here, B_{max} is the maximum bitrate limited by the application (set as 50 *Mbps* in our implementation). τ denotes the accumulated number of received frames after initialization (discussed later). This equation is designed to be inversely related to the current bitrate B, to enhance fairness convergence by increasing the value of I more slowly for flows with higher bitrates.

Since the value of I keeps increasing, Pudica resets τ to be zero when $\widetilde{R} > 1$, in order to prevent a heavy queuing caused by an overlarge I. This initialization method relies on BUR estimations, which may not be flawless. Thus, Pudica further implements a *time-driven* initialization mechanism to avoid severe unfairness. To be specific, τ is initialized every five seconds. This periodical and highly synchronized τ initialization enhances the robustness of our AI-MD scheme. By applying AI-MD with an adaptive AI step, Pudica can achieve faster convergence to fairness while avoiding frequent queue buildups (further illustrated in Appendix C).

4.3 Bitrate Adaptation over the Short-Term BUR

Using a smoothed BUR estimation that combines multiple BUR samples from the past can provide more robustness to network noises, compared to relying solely on a single-frame BUR. However, due to its lagging nature, the smoothed BUR may not be able to respond promptly to improper control decisions or sudden network changes. To address this issue and achieve more timely adaptations, we propose two additional bitrate adjustment approaches that utilize the short-term BUR, particularly the BUR of recently received frames. By combining the responsiveness of short-term BUR with the robustness of smoothed BUR, Pudica strikes a superior balance between timeliness and stability in bitrate control.

Temporary bitrate fallback. Upon detecting a potential overshoot or congestion *for the first time*, Pudica instantly takes action by temporarily reducing the bitrate. Specifically, when the BUR feedback of any single frame exceeds one, Pudica will reduce the bitrate by ζ (set to be 15%, empirically) for the subsequent frame to be encoded. This bitrate reduction is temporary, meaning that after adjusting the bitrate for the next to-be-encoded frame, the encoder will revert back to the previous bitrate setting.

It is important to highlight that in cases of sudden congestion, the sender may experience delays in receiving feedback regarding network degradation, which can lead to delayed bitrate fallback. To address this issue, we monitor the frames that have been sent to the network but have not yet been acknowledged. We introduce the concept of *next delay*, which represents the elapsed time *from* the moment the next to-be-received frame (i.e., the earliest sent frame among the in-flight frames) was sent *until* the current time. When the next delay is significant, Pudica performs the bitrate fallback as well. The introduction of the next delay signal enables Pudica to respond to congestion more swiftly and timely.

This fallback scheme could reduce queue buildups caused by potential overshoots since we acknowledge the inherent difficulty in achieving flawless BUR predictions. The transient nature of the bitrate reduction helps minimize the impact of incorrect fallback due to jitter-induced delay variations.

Active queue draining. When the BUR feedback of three recently received frames *both* exceeds one, Pudica enters the queue-draining phase with active undershooting. At this time, the bitrate is set to be:

$$B^{new} = \alpha \times receiving\_rate - draining\_rate, \qquad (11)$$

where *receiving_rate* is the average data receiving rate from the onset of congestion to the present moment. It serves as a reliable estimation of the maximum throughput achievable during this period. Additionally, Pudica employs a calculation to determine the required additional throughput rates (referred to as *draining_rate*) needed to clear the self-induced queuing at the bottleneck within the next 200 *ms*. Here, the volume of self-induced queued data is quantified by measuring the number of in-flight packets. By carrying out an active queue draining over the Eq. 11, Pudica is able to efficiently and swiftly empty the existing queue at the bottleneck.

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >100/200ms**	**Avg. bitrate**
Copa	26.9ms	47.2/79.9ms	1.68%/0.77%	42.9Mbps
Salsify	21.5ms	30.0/67.0ms	0.48%/0.09%	42.4Mbps
SQP	41.8ms	101.7/147.8ms	1.27%/1.03%	43.7Mbps
Pudica	**19.5ms**	**25.0/30.6ms**	**0.07%/0.004%**	**47.5Mbps**

Table 1: Overall system-level performance at scale (**Ethernet**).

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >100/200ms**	**Avg. bitrate**
Copa	76.5ms	305.2/672.8ms	7.7%/3.9%	22.6Mbps
Salsify	195.5ms	631.7/982.5ms	22.9%/13.5%	**32.6Mbps**
SQP	302.2ms	815.2/1218.5ms	10.3%/8.7%	23.2Mbps
Pudica	**33.7ms**	**74.9/175.6ms**	**2.5%/0.72%**	31.2Mbps

Table 2: Overall system-level performance at scale (**WiFi**).

On the other hand, when the congestion vanishes, gradually increasing the bitrate from a low value can lead to suboptimal utilization due to the delayed nature of smoothed BUR estimation. Therefore, when the BUR of a recently arrived frame is less than one, Pudica recalculates the current *receiving_rate* and directly restores the bitrate to match it. Considering the delay sensitivity of the queue draining mechanism, this one-step recovery scheme improves resilience to network jitters and consequently enhances link utilization, by promptly recovering when the delay returns to normal.

5 Evaluation

We deployed Pudica and the state-of-the-art CC algorithms on a commercial cloud gaming platform, and conducted extensive evaluations in real-world wired and wireless networks. We also evaluated the ability of convergence to efficiency and fairness through a large-scale, in-the-wild network testing platform. We demonstrated the reason for Pudica's gain by miscellaneous emulation experiments. Finally, we proved the effects of several selected algorithm components on end-to-end performances by microbenchmarks. All timeline-based line charts in this section are presented at a frame-wise level.

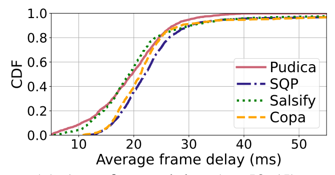
5.1 Methodology for Large-Scale Algorithm Evaluation

We deployed four CC approaches, including Pudica, Salsify [13], Copa [11], and SQP [20] on Tencent START cloud gaming platform [9] for large-scale A/B tests[3]. The START system integrates the customized network protocol stack and video codec, supporting submillisecond-class pacing, packet-level acknowledging, and frame-level size control.

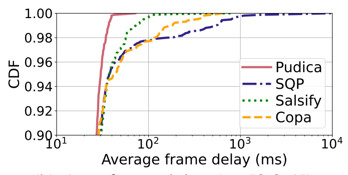
Experiment setups. Our commercial system operates on fully public networks over the Internet. The evaluation finally involved more than 57,000 gaming sessions across 15 cities, two network types (Ethernet and WiFi), and three ISPs over five weeks. We randomly chose one of the CC algorithms for each session and kept the other system modules the same for a fair comparison. We set the frame rate as 60 and the maximal bitrate as 50 Mbps[4] for all algorithms. We set

[3]A preliminary emulation experiment is conducted in Appendix D.

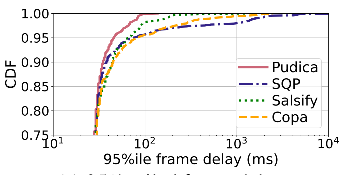
[4]In production cloud gaming services, the gaming bitrate above 50 Mbps is bandwidth-costly with low marginal benefits on player QoE.



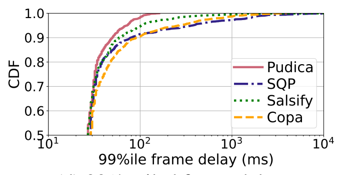
(a) Avg. frame delay (y∼[0-1]).



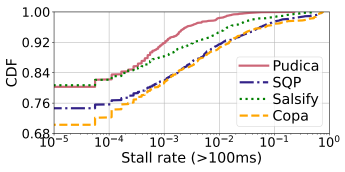
(b) Avg. frame delay (y∼[0.9-1]).



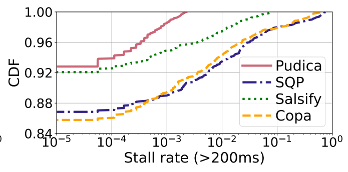
(c) 95% tailed frame delay.



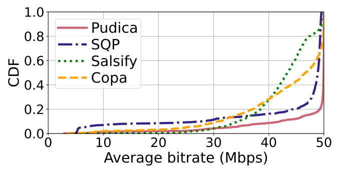
(d) 99% tailed frame delay.



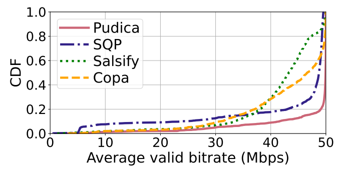
(e) Stall rate (>100ms).



(f) Stall rate (>200ms).

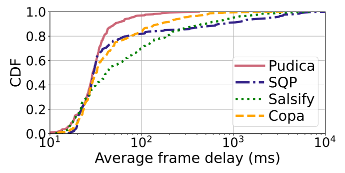


(g) Average bitrate.

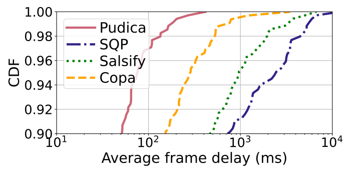


(h) Average valid bitrate.

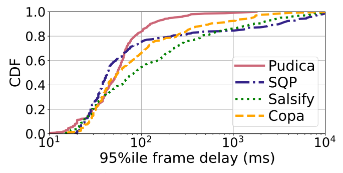
Figure 8: Performance at scale on a commercial cloud gaming platform over the Internet (**Ethernet**). X-axis in some figures is in log scale.



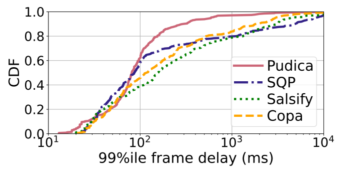
(a) Avg. frame delay (y∼[0-1]).



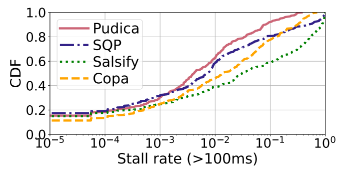
(b) Avg. frame delay (y∼[0.9-1]).



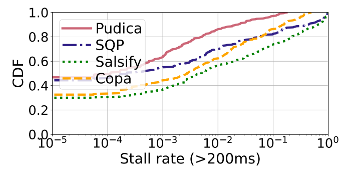
(c) 95% tailed frame delay.



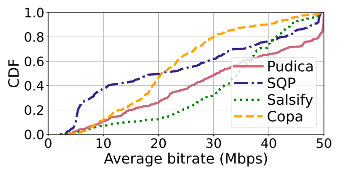
(d) 99% tailed frame delay.



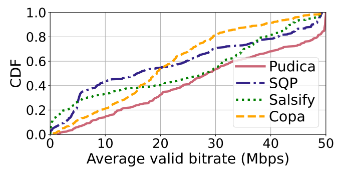
(e) Frame stall rate (>100ms).



(f) Frame stall rate (>200ms).



(g) Average frame bitrate.



(h) Average valid bitrate.

Figure 9: Performance at scale on a commercial cloud gaming platform over the Internet (**WiFi**). X-axis in some figures is in log scale.

the delay constraint as $(L+D_{min})$ in Salsify. Following [20], Copa was implemented with a setting of $\delta = 0.1$ and without mode switching. For CWND-based methods, the frame bitrate was determined based on the ratio of the CWND size to the smoothed RTT.

Evaluation metrics. To evaluate operation interactivity, we computed several metrics for each gaming *session*, including the average, 95%ile, and 99%ile round-trip frame delay, and the stall rates for frame delays exceeding 100 *ms* and 200 *ms*. For display quality, we computed the average values of frame bitrate and valid bitrate for each session. The *valid bitrate* refers to the average bitrate of frames that have a low delay, specifically less than 50 *ms*. While a delayed frame may have a high bitrate, the information it carries could be outdated and therefore useless to the player experience.

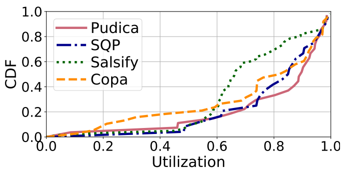
5.2 System-Level Performance at Scale

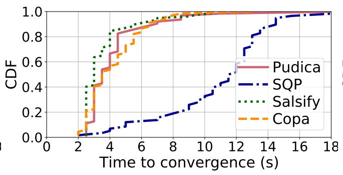
We list evaluation results by averaging across all sessions in Tab. 1 and Tab. 2, and we plot the CDF curves of the various metrics in Fig. 8 and Fig. 9. In summary: *i)* Pudica significantly reduces frame delay and stall rate, and cuts the delay tail. *ii)* Pudica achieves an equivalent or higher bitrate. *iii)* Pudica strikes a better balance between delay and bitrate.

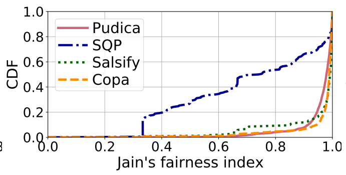
Average and tailed frame delay. Fig. 8a–8d illustrates that compared to the baselines, Pudica reduces the frame-level average delay, 95%ile delay, and 99%ile delay by 1.5×, 2.4×, and 3.2×, respectively, over Ethernet networks. Similarly, over WiFi networks (Fig. 9a–9d), the corresponding reductions are 5.7×, 7.8×, and 5.5×, respectively. We observed that Salsify experienced lower delays in situations of highly underutilized bandwidth (especially on wired networks) due to its relatively bursty sending, as compared to methods with conservative pacing control. However, Salsify exhibited excessive sensitivity to intensive packet arrival or jitters, which are common occurrences in WiFi networks, leading to dramatic overshoots, high queuing delays, and frequent stalls.

Stall rate. Compared to the alternatives, Pudica reduces the stall rate with thresholds of 100 *ms* and 200 *ms* by 16.3× and 22.5×, respectively, over Ethernet networks (Fig. 8e–8f). On WiFi networks (Fig. 9e–9f), the reductions are 5.5×–12.1×. We noticed that many users had access to sufficiently high bandwidth, and as a result, experienced negligible stall rates regardless of the CC algorithm employed, due to the upper limit of 50 *Mbps* set in our experiments. Nonetheless, there is still a notable proportion of sessions that can derive benefits from the implementation of Pudica.

Frame bitrate. As depicted in Fig. 8g and 9g, Pudica achieves a 1.10× and 1.19× bitrate enhancement on average over the Ethernet and WiFi networks, respectively. While Salsify outperforms Pudica in average bitrate for WiFi scenarios, its average delay and stall rate is 5.8× and 9.2× higher than Pudica, respectively. From Fig. 8h and Fig. 9h, we can


(a) Avg. utilization.


(b) Convergence speed (efficiency).


(c) Fairness.

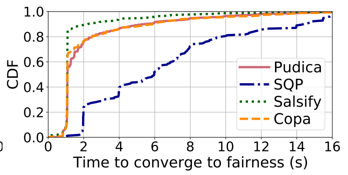

(d) Convergence speed (fairness).

Figure 10: Convergence ability evaluation for efficiency and fairness through a large-scale, in-the-wild network testing platform [24].

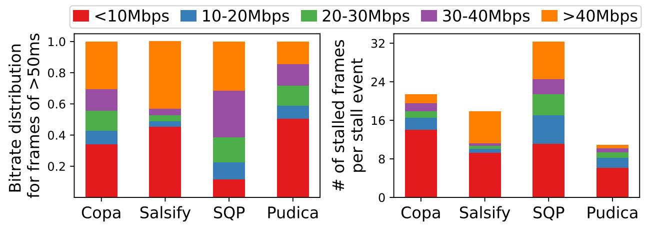


Figure 11: The proportion of different bitrates for frames with a delay greater than 50 *ms* (**Left**), and the average number of consecutively stalled frames per occurrence of stall event (**Right**).

see that Pudica achieves the highest valid bitrate over both Ethernet and WiFi networks.

Deeper analysis. To gain a better understanding of how Pudica reduces the stall rate and cuts the delay tail, we focused on frames with large delays and observed how these CC schemes respond to stalls. In Fig. 11, we present the bitrate distribution for frames with a delay exceeding 50 *ms* (*left-side*), and the number/bitrate of consecutively stalled frames per occurrence of stall event (*right-side*). Here, we set a threshold of 100 *ms* for judgment of stall events/frames. Our analysis shows that for Pudica, low-bitrate (<10 *Mbps*) frames make up more than half of frames with a delay exceeding 50 *ms* (similar results for consecutively stalled frames), indicating Pudica's superior performance in reducing CC-induced queuing. When a stall event occurs, Pudica can quickly respond to it by reducing the bitrate, resulting in a significant reduction in the average stall count to 10.9 frames.

5.3 In-the-Wild Evaluation for Efficiency and Fairness

We used a large-scale network test platform [24] (detailed in Appendix E) to evaluate the efficiency and fairness of various CC algorithms, as well as the speed at which they reached "steady state" and "fair state". A flow is deemed to be in a steady state if it does not experience a bitrate change exceeding 2% within the last two seconds. The fair state is defined as a situation wherein the mean throughput of any two flows does not diverge by more than 20% within a two-second interval, or both flows have reached steady states.

Efficiency. Efficiency and its convergence are measured by bandwidth utilization and the duration necessitated for each algorithm to attain the steady state. Each test involved measuring available bandwidth over a 90-second span, followed by the random selection and execution of one CC algorithm for 60 seconds. The average bandwidth utilization of the last 30 seconds of each test was calculated, as shown in Fig. 10a. The findings indicate that Pudica achieved an average utilization of 77.6%, while SQP, Copa, and Salsify achieved 76.8%, 70.8%, and 68.7%, respectively. Fig. 10b plots the time to reach a steady state. The majority of flows (>80%) in Pudica and Salsify converged within 6 seconds, whereas SQP required an average of 9.7 seconds to reach a steady state.

Fairness. Fairness and its convergence are measured using Jain's fairness index and the time necessitated for each algorithm to reach the fair state. In each test, we initiated one flow, followed by the second and third flows with the same algorithm after 20 and 40 seconds, respectively. Jain's fairness index was calculated using the throughput data collected during the 45-60 second interval, as plotted in Fig.10c. Pudica achieved an average fairness index of 0.95, while SQP, Copa, and Salsify achieved 0.735, 0.965, and 0.947, respectively. Fig. 10d illustrates the time to reach the fair state. We can see that, Salsify achieved the fastest fairness convergence, while SQP required the longest time to converge to a fair state.

Summary. Pudica provides high utilization and good fairness, while also achieving rapid convergence to both. Consequently, Pudica strikes a better balance between efficiency, fairness, and low latency, compared to the baselines.

5.4 Pudica Deep Dive over Emulation

We assessed Pudica's convergence capability over the Mahimahi emulation [37], elucidating the reason for improvement.

Consistent convergence to low-queuing efficiency. As depicted in Fig. 12, Pudica exhibits a faster and more stable adaptation to network variations, both in terms of bandwidth decrease and increase, compared to the baselines. When the bandwidth abruptly reduces, Pudica employs the active queue draining mechanism (§4.3) to trigger a dramatic bitrate fallback in response to congestion. This allows Pudica to quickly empty the queue, a process that typically takes only 200 *ms*. Once the queue has been emptied, Pudica recovers the bitrate by one step, ensuring a reasonable level of link utilization. This strategy also provides Pudica with robustness against network jitters, as demonstrated in Appendix F. As a consequence, Pudica achieves a superior balance between low queuing and efficiency. On the other hand, SQP exhibits slow adaptation to reduced bandwidth, taking nearly five seconds to drain the queue, resulting in high delay spikes and prolonged stall duration. Salsify and Copa are badly sensitive to both transient packet queuing and empty queues, causing choppy rate decisions and persistent delay oscillations.

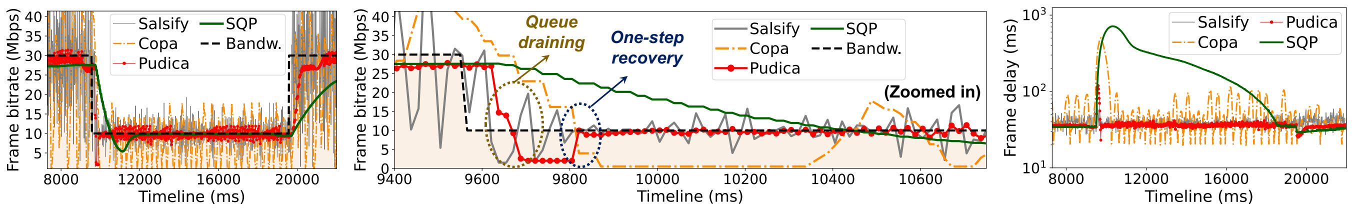
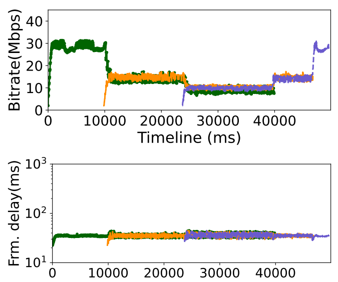
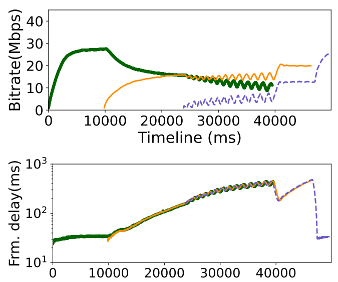


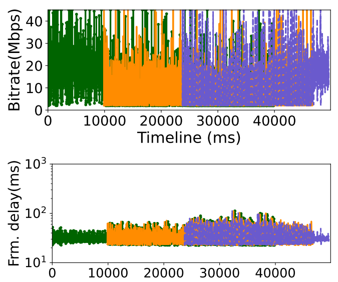
Figure 12: Comparison of convergence to low-queuing efficiency: bitrate and delay of each frame as the bandwidth alters. Pudica reconciles the low queuing delay and high bandwidth utilization by merging responsive queue draining and rapid, stable bandwidth approaching.



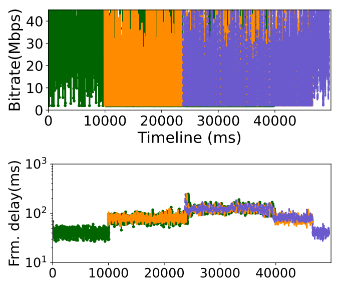
(a) Pudica



(b) SQP



(c) Copa



(d) Salsify

Figure 13: Comparison of fairness convergence: bitrate and delay of each frame for multiple competing flows as they gradually enter and quit. Up to three flows share a 30 *Mbps* bottleneck. Pudica achieves rapid, stable convergence to fairness with sustaining low frame delays.

Rapid and stable convergence to fairness. As shown in Fig. 13, Pudica achieves rapid and consistent fairness convergence for both throughput and delay whenever a new stream enters or leaves the same link. Pudica also prevents the formation of significant queues in the presence of competing flows. SQP utilizes an essentially MIMD-based rate adjustment approach, leading to poor fairness, especially when a slow-start newcomer encounters multiple existing high-bitrate flows. Further, multiple concurrent SQP flows would mistakenly regard each other as the buffer-filling flow and enter the competing mode by continually raising the estimate of minimum delay, resulting in an increasing delay. While Salsify and Copa achieve decent fairness, they suffer from significant bitrate and delay oscillation when multiple flows are present.

5.5 Microbenchmark

We conducted a microbenchmark in real networks to quantify the contribution of different design elements to overall performance. We plot the key evaluation results in Fig. 14 and provide more detailed information in Appendix G.

Different BUR estimation methods. We replaced the BUR estimator in Pudica with the network estimation methods introduced in SQP [20] and Salsify [13]. SQP uses the frame as a burst to probe the available bandwidth, while Salsify utilizes packet trains to probe the available capacity. Therefore, these methods do not directly estimate the BUR indicator. That being said, we evaluated the performance of their probing methods within the Pudica framework. To accomplish this, we considered the ratio of the current bitrate to the estimated available bandwidth/capacity as the BUR estimation in the Pudica variants. As depicted in Fig. 14a, Pudica achieves lower stall rates and tailed delay, particularly in multi-flow scenarios, when compared to the variants. This confirms the advantages of our BUR estimation approach.

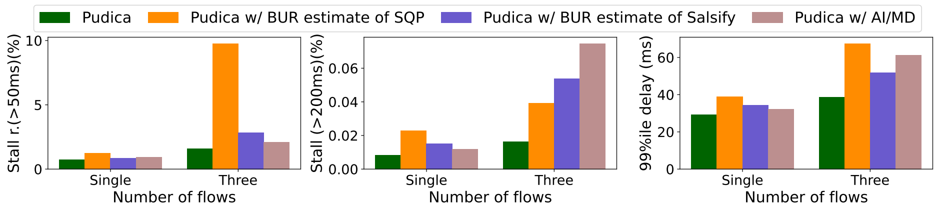
AI-MD v.s. AI/MD. Fig. 14a also compares the strategy of simultaneous AI and MD in Pudica (i.e., AI-MD) with the traditional AI/MD method. For AI/MD, we additively increase the bitrate when the BUR is between α and one; we multiplicatively decrease the bitrate when the BUR exceeds one. Our findings demonstrate that Pudica with AI/MD achieves comparable performance to AI-MD in single-flow scenarios. However, when three flows operate simultaneously on the same link, AI-MD exhibits the advantage of reducing stall rate and tailed delay (as well as fairness, see Appendix G).

Temporary bitrate fallback and active queue draining. Fig. 14b evaluates the contributions of two bitrate adjustment methods driven by short-term BUR signals (§4.3). Evidently, the introduction of temporary fallback and active queue draining significantly improves the stall rate and tailed delay.

Sensitivity analysis for algorithm parameters. We assessed the impacts of several parameter choices in Pudica. Tab. 6 in the Appendix shows that Pudica maintains consistent performance across various parameter configurations, indicating its lesser dependence on specific parameter settings.

6 Discussion

Granularity for periodic bursts. Pudica uses a network probing approach where all packets within a frame are treated as a burst, minimizing sender-side delay and aligning with our focus on frame-level end-to-end delay. This granularity is suitable for our context. However, periodic burst transmission at different granularities could also be effective, especially for applications beyond frame-level performance assessment.



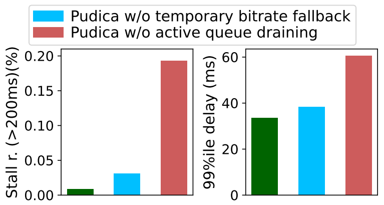


Figure 14: Key microbenchmark results: (a) Different BUR estimation; AI-MD v.s. AI/MD. (b) Responsive reaction.

Balancing feedback usefulness and cross-traffic detection. Pudica may intentionally slow the pace multiplier to boost the chance of detecting competing flows. While this packet pacing may reduce feedback usefulness by eliminating packet queuing, it shortens the agnostic period and lessens BUR underestimation due to cross flow unawareness. This is a trade-off between feedback precision and flow detection. Our method may overrate BUR compared to fully bursty transmission, impacting efficiency convergence. However, it improves delay performance, a more fitting trade-off for cloud gaming.

Handling packet losses. Since Pudica estimates available bandwidth based on the actual arrived size of frames, it would inherently respond to packet losses, triggering bitrate decreases, due to the reduced amount of arrived packets when losses occur. However, the nature of the frame-as-a-whole sending pattern in video streaming, coupled with various types of network bottlenecks, can still result in losses despite BUR being below 100% (e.g. in shallow buffer scenarios), causing additional frame delays and unnecessary bitrate downgrades. Moving forward, our research aims to explore practical strategies for mitigating the impact of packet losses through both proactive and reactive ways.

Competitiveness with buffer-filling flows. Pudica achieves a decent fairness for *homogeneous* flows. However, unlike mode-switched CC methods (e.g., Copa [11], Nimbus [33]), Pudica does not explicitly compete with buffer-filling or *inelastic* flows. Instead, Pudica concedes to them by lowering the bitrate whenever the queue becomes full. This approach is adopted because sacrificing delay for high bitrate is meaningless for cloud gaming. Actually, Pudica keeps fair competitiveness with loss-based flows under the circumstance of *shallow* bottleneck buffers (see Appendix H).

User-centric QoE optimization. Pudica primarily aims to achieve the highest possible bitrate while maintaining low latency. However, it may not necessarily result in the optimal user QoE. For instance, excessively high bitrates may only marginally enhance video quality but can make the streaming session more susceptible to bandwidth fluctuations. In the future, we will explore the opportunity that leverages cross-layer and user-centric approaches (e.g., introducing visual quality instead of bitrate) to further enhance player QoE.

Potential gains for other low-latency applications. While Pudica is designed for cloud gaming, we believe that the concepts and principles embedded within our CC framework could be beneficial for other networked applications that require both low latency and high throughput. We intend to investigate its potential across various domains.

7 Other Related Work

Prior works [22, 23, 35, 38] utilize explicit signals to judge the precise level of utilization or congestion, enabling convergence to both low queuing, efficiency, and fairness. However, it remains challenging to deploy these methods at scale. Nimbus [33] tries to detect the elasticity of cross traffic and dynamically switches between throughput-competitive and delay-controlling modes based on its detection. It still prioritizes throughput over delay, which is unable to satisfy the need for consistent low delay. Several works explore the use of deep reinforcement learning on bitrate adaptation for video telephony [14–16]. They partly imitate the behavior of GCC agents or take GCC as the protective backup, thereby with similar limitations as GCC. Moreover, cross-flow fairness is not carefully handled by these methods. A detailed comparison between our network probing method and traditional packet-train techniques is included in Appendix I.

8 Conclusion

We present Pudica, a practical congestion control (CC) algorithm designed for cloud gaming systems. Given a constraint of near-empty bottleneck queues, we rethink how to achieve efficiency and fairness in end-to-end CC design, regardless of single-flow or multiple-flow scenarios. To reach this goal, we propose a novel network probing method to estimate the bandwidth utilization ratio (BUR) of the bottleneck link. By leveraging both long-term and short-term BUR estimations, we design several intuitive but effective control strategies to minimize the queuing delay while maintaining efficiency and fairness. By conducting large-scale experiments on Tencent START cloud gaming services in both wired and wireless networks, we demonstrate that Pudica considerably reduces the frame delay and stall rate while preserving high bitrate and decent fairness, compared to the alternatives.

Acknowledgements: We sincerely thank our anonymous reviewers for their valuable comments. This work was supported by Tencent Rhino-Bird Research Elite Program, National Key Research and Development Program of China under Grant 2021YFB2401300, and National Natural Science Foundation of China under Grant U21A6005. Shusen Yang, Jing Wang, and Honghao Liu are the corresponding authors.

References

[1] Hao Chen, Xu Zhang, Yiling Xu, Ju Ren, Jingtao Fan, Zhan Ma, and Wenjun Zhang. T-gaming: A cost-efficient cloud gaming system at scale. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2849–2865, 2019.

[2] Xu Zhang, Hao Chen, Yangchao Zhao, Zhan Ma, Yiling Xu, Haojun Huang, Hao Yin, and Dapeng Oliver Wu. Improving cloud gaming experience through mobile edge computing. *IEEE Wireless Communications*, 26(4):178–183, 2019.

[3] Pouya Hamadanian, Doug Gallatin, Mohammad Alizadeh, and Krishna Chintalapudi. Ekho: Synchronizing cloud gaming media across multiple endpoints. In *SIGCOMM*, pages 533–549, 2023.

[4] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *SIGCOMM*, pages 710–723, 2023.

[5] Sandeepa Bhuyan, Shulin Zhao, Ziyu Ying, Mahmut T Kandemir, and Chita R Das. End-to-end characterization of game streaming applications on mobile platforms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–25, 2022.

[6] Amazon. Amazon luna: Amazon's cloud gaming service. https://luna.amazon.com/.

[7] Microsoft. Xbox cloud gaming (beta) on xbox.com. https://www.xbox.com/en-us/play.

[8] NVIDIA. Your games. your devices. play anywhere | nvidia geforce now. https://www.nvidia.com/en-us/geforce-now/.

[9] Tencent. START Cloud Gaming. https://start.qq.com/.

[10] Grand View Research. Cloud gaming market size, share & trends analysis report. https://www.grandviewresearch.com/industry-analysis/cloud-gaming-market.

[11] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *NSDI*, pages 329–342, 2018.

[12] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking*, 25(5):2629–2642, 2017.

[13] 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 *NSDI*, pages 267–282, 2018.

[14] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *MOBICOM*, pages 1–16, 2019.

[15] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *MOBICOM*, pages 1–14, 2020.

[16] Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models. In *MOBICOM*, pages 775–788, 2021.

[17] Xiaoqing Zhu and Rong Pan. Nada: A unified congestion control scheme for low-latency interactive video. In *International Packet Video Workshop*, pages 1–8. IEEE, 2013.

[18] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[19] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. Pcc vivace: Online-learning congestion control. In *NSDI*, pages 343–356, 2018.

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

[21] Wikipedia. Mimosa pudica. https://en.wikipedia.org/wiki/Mimosa_pudica.

[22] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*, pages 89–102, 2002.

[23] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. One more bit is enough. In *SIGCOMM*, 2005.

[24] Bonree. Data - bonree one. https://www.bonree.com/.

[25] 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 *NSDI*, 2023.

[26] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. *SIGCOMM CCR*, 41(4):362–373, 2011.

[27] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[28] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006.

[29] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, pages 459–471, 2013.

[30] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *SIGCOMM*, 43(4):123–134, 2013.

[31] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. Pcc: Re-architecting congestion control for consistent high performance. In *NSDI*, pages 395–408, 2015.

[32] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *SIGCOMM*, pages 509–522, 2015.

[33] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. Elasticity detection: A building block for internet congestion control. In *SIGCOMM*, pages 158–176, 2022.

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

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

[36] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. *SIGCOMM CCR*, 32(4):295–308, 2002.

[37] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *ATC*, pages 417–429, 2015.

[38] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. ABC: A simple explicit congestion controller for wireless networks. In *NSDI*, pages 353–372, 2020.

[39] Xinlei Yang, Xianlong Wang, Zhenhua Li, Yunhao Liu, Feng Qian, Liangyi Gong, Rui Miao, and Tianyin Xu. Fast and light bandwidth testing for internet users. In *NSDI*, pages 1011–1026, 2021.

[40] SpeedTest. The global broadband speed test. https://www.speedtest.net/.

[41] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[42] 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.

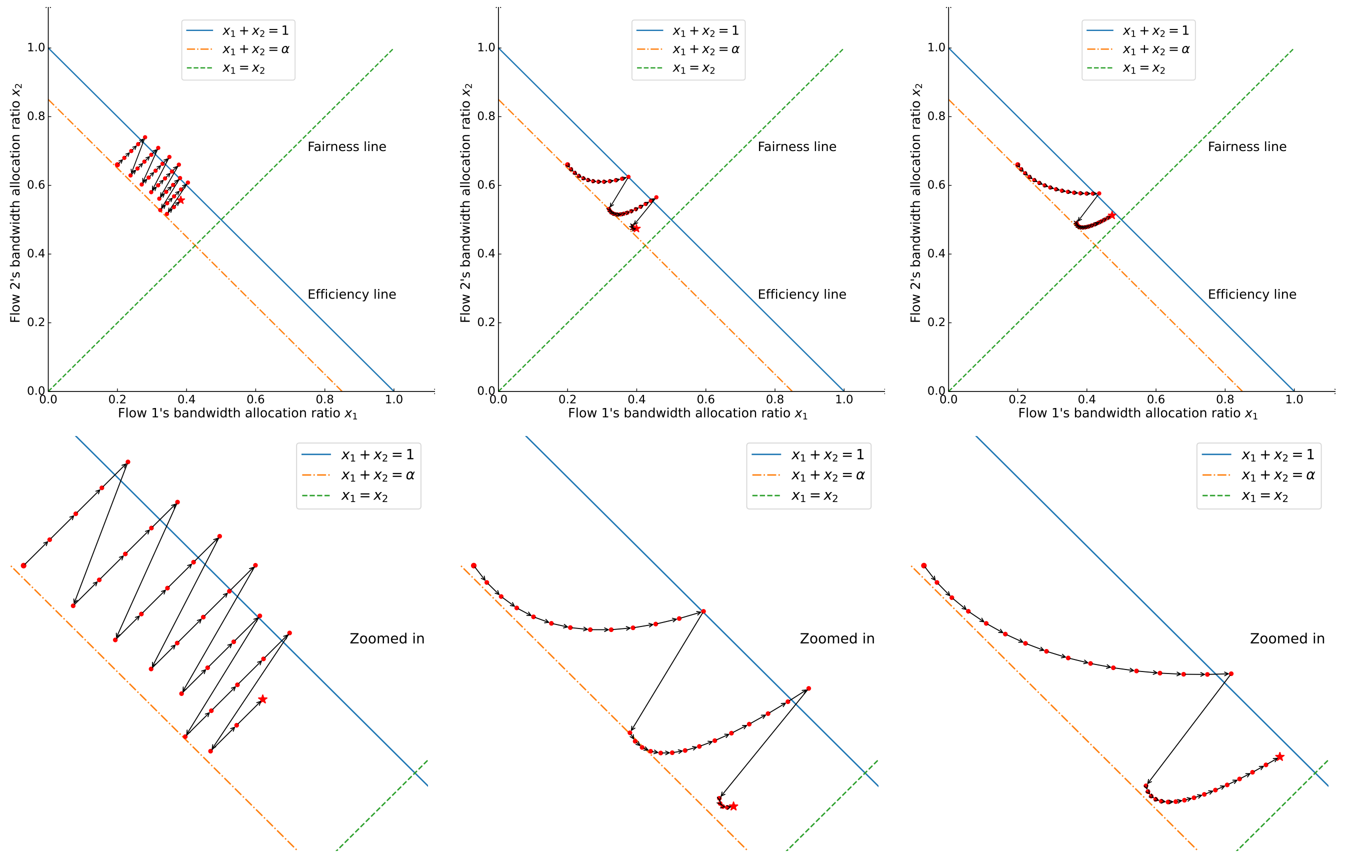
[43] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, 2004.

[44] Ravi Prasad, Constantine Dovrolis, Margaret Murray, and Kimberly Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, 2003.

[45] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *INFOCOM*, volume 2, pages 905–914. IEEE, 2001.

A Validation Methodology for BUR Estimation Methods

In this section, we present the methodology used to validate the effectiveness of our BUR estimation method. Following [39], we regard the estimated bandwidth obtained through a flooding-based probing approach (similar to SpeedTest [40]) as the ground truth of bottleneck bandwidth, denoted by C. For each round of tests, after a short period of probing to judge C, we conducted two simultaneous flows on the same connection.



(a) Traditional AI/MD (b) AI-MD w/ bitrate-independent adaptive AI (c) AI-MD w/ bitrate-dependent adaptive AI

Figure 15: Fairness convergence trajectory for the traditional AI/MD paradigm and our AI-MD method (i.e., simultaneous AI and MD). Here, these diagrams are presented for illustration purposes and do not exactly align with the proposed algorithms in Pudica.

One flow adopted our probing strategies and was set to a fixed bitrate B. The other flow acted as the competing flow with a *random* pace multiplier (to simulate complex traffic patterns) and the same bitrate B. To evaluate the performance of various BUR estimation methods, we use the ratio $(2 \times B)/C$ as the actual BUR. This allowed us to assess how well the estimation methods performed in estimating the true BUR.

B Sample Weights for Smoothed BUR Computation

Due to the interdependence of sending behaviors and network feedback, we assign different weights to the BUR samples to enhance estimation robustness. Specifically, we set:

$$\omega_k^{I} = min\big(\, R_k + 1,\ 2\,\big), \tag{12}$$

$$\omega_k^{II} = min\big(\, B_k + 10,\ 50\,\big), \tag{13}$$

$$\omega_k^{III} = k + 20, \tag{14}$$

$$\omega_k = \frac{\omega_k^{I} \times \omega_k^{II} \times \omega_k^{III}}{\sum_{j=1}^{N_{packet}} \omega_j^{I} \times \omega_j^{II} \times \omega_j^{III}}. \tag{15}$$

ω_k symbolizes the importance weight given to the k-th BUR sample, comprising three elements. ω_k^{I} assigns higher importance to samples with longer frame delays, as they spend more time in flight. ω_k^{II} assigns higher importance to samples with larger frame sizes, as they exhibit greater estimation robustness against delay jitters. ω_k^{III} assigns higher importance to more recent samples, as they provide fresher feedback.

C Fairness Convergence Trajectory for Different AI-and-MD Schemes

As referenced in [41], we plotted the trajectories of a two-flow system starting from the same point using various AI-and-MD policies via numerical simulation, which is depicted in Fig. 15. On the figures, the horizontal axis represents the bandwidth allocation ratio of Flow 1 (denoted by x_1), while the vertical axis corresponds to Flow 2 (denoted by x_2). Allocations satisfying $x_1 + x_2 = 1$ indicate efficient allocations, representing 100% bandwidth utilization (refer to *Efficiency line*). Allocations satisfying $x_1 = x_2$ represent fair allocations (refer to *Fairness line*). The optimal point is the intersection of these two lines. The objective of control schemes should be to converge the system to this optimal point, irrespective of the initial position.

In the AI/MD method (see Fig. 15a), we additively increase the bitrate when the BUR exceeds α but remains below one; we multiplicatively decrease the bitrate by 15% when the BUR surpasses one. For AI-MD methods, Fig. 15b illustrates

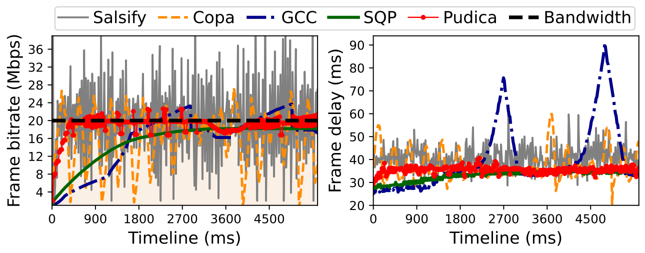


Figure 16: Preliminary validation: the performance of various baseline algorithms using Mahimahi emulations with a plain trace featuring a constant bandwidth of 20 *Mbps*.

the trajectory of AI-MD with a bitrate-independent adaptive AI scheme, where the AI step (i.e., I) is linearly increased. In contrast, the AI-MD method in Pudica (see Fig. 15c) adopts a bitrate-dependent adaptive AI scheme, where the magnitude of increased AI steps (i.e., ΔI) is inversely proportional to the bitrate. Compared to the AI/MD method, AI-MD reduces the occurrences of over-sending and thus lowers the average queuing delay, by executing MD before queue construction. At the same time, AI-MD speeds up fairness convergence by increasing the frequency of throughput reallocation. Moreover, the implementation of bitrate-dependent AI step adaptation further enhances the convergence speed toward fairness.

D Preliminary Evaluation for Various CC Algorithms through Emulation

Prior to conducting large-scale experiments over the Internet, we evaluated Pudica and the other potential solutions (i.e., GCC [12], Copa [11], Salsify [13], and SQP [20]) on the Mahimahi network emulator [37]. These evaluations were performed using a trace that maintained a constant bandwidth of 20 *Mbps* and RTTs of 20 *ms*. As depicted in Fig. 16, Pudica and SQP achieve consistent bandwidth convergence with negligible queuing delays. Salsify and Copa maintain low frame delays in spite of significant oscillations in bitrates. In contrast, GCC fails to consistently achieve low latency, displaying periodic delay spikes even under a constant bandwidth. Therefore, we selected Pudica, Salsify, Copa, and SQP for further performance testing at scale (shown in §5.2).

E A Large-Scale Dummy Client Platform

In our study, we utilized a specialized dummy client platform called Bonree [24], instead of the real-user platform (i.e., START [9]), to evaluate the convergence ability of efficiency and fairness in the wild. We opted for the Bonree platform due to the additional client privileges required for testing the aforementioned metrics, such as the permission to run multiple streams simultaneously. These privileges cannot be accommodated by our profit-oriented cloud gaming services.

Bonree offers millions of end devices (e.g., PCs) distributed globally to emulate real user clients. These end devices possess computing capacities and network resources similar to those commonly used by consumers. They can install customized software, e.g., the cloud gaming client app, and establish connections with the cloud gaming servers designated by us over the Internet. As a result, the gaming server can transmit frame streams to these end devices and receive their feedback. Additionally, these devices can record the necessary logs as per our requirements. Unlike real users, we can impose more demands on these devices without concerns about user experience. Thus, we employed Bonree to conduct experiments regarding efficiency and fairness convergence. Note that the end devices on Bonree communicate with servers through fully public networks. Consequently, the information within the network remains unknown.

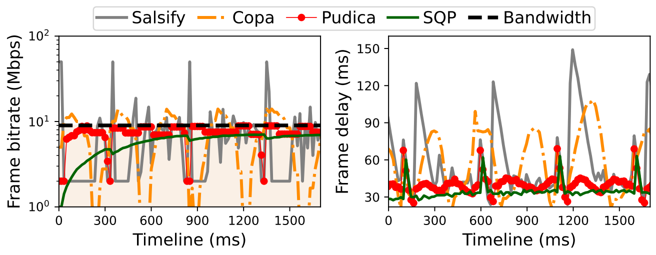


Figure 17: Robustness test with a 40 *ms* jitter per period of 500 *ms*. Pudica achieves a good balance between fast responsiveness to potential congestion and jitter resilience.

F Pudica Robustness to Network Jitters

In Fig. 17, we assessed Pudica's robustness to delay jitters by introducing a 40*ms* jitter every 500*ms* period. While this jitter delays some packet arrivals, it does not impact the bandwidth. These delayed packets eventually reach the receiver with subsequent packets, leading to a burst arrival. The results show that Pudica effectively balances sensitivity to high delay and resilience against jitters. While SQP shows strong robustness to jitters, it struggles with slow bandwidth adaptation (Fig. 12). The burst packet arrival can mislead Salsify and Copa to momentarily overrate network conditions, which causes bitrate overshoots and high delay spikes.

G Additional Microbenchmark Results

This section presents more statistics from our microbenchmark experiment. Tab. 3 and Tab. 4 showcase the performances of multiple Pudica variants under a single flow and three simultaneous flows, respectively. These experimental results confirm the efficacy of Pudica's BUR estimation method and AI-MD technique. For Pudica with AI/MD, we substitute our AI-MD scheme with the sole additive increase when $\alpha < BUR \leq 1$. MD operations will be triggered by our queue-draining scheme when $BUR > 1$. Tab. 5 shows the effectiveness of Pudica's control schemes driven by short-term BUR signals in reducing tail delays and stall rates. Tab. 6 presents Pudica's performance under various parameter settings.

H Competitiveness with Buffer-Filling Flows

Figure 18 depicts the Pudica performance when it encounters a buffer-filling flow. Upon initiating a Pudica flow, we subsequently launched and terminated a Cubic flow [42] at 5s and

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >50/100/200ms**	**Avg. bitrate**
Vanilla Pudica	18.0ms	22.3/29.2ms	0.73%/0.09%/0.0083%	46.940Mbps
Pudica w/ BUR estimation of SQP	20.1ms	26.1/39.0ms	1.25%/0.21%/0.0229%	47.103Mbps
Pudica w/ BUR estimation of Salsify	18.4ms	22.9/34.2ms	0.84%/0.13%/0.015%	41.164Mbps
Pudica w/ AI/MD	18.3ms	22.7/32.2ms	0.91%/0.10%/0.0117%	46.181Mbps

Table 3: Microbenchmarks: the performance of different Pudica variants when launching a single flow.

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >50/100/200ms**	**Avg. bitrate**	**Fairness index**
Vanilla Pudica	21.5ms	29.2/38.6ms	1.60%/0.11%/0.0164%	31.962Mbps	0.958
Pudica w/ BUR estimation of SQP	31.9ms	49.2/67.5ms	9.77%/0.42%/0.0391%	33.132Mbps	0.952
Pudica w/ BUR estimation of Salsify	21.7ms	34.2/51.8ms	2.81%/0.25%/0.0538%	33.185Mbps	0.941
Pudica w/ AI/MD	22.3ms	32.8/61.2ms	2.10%/0.29%/0.0747%	32.209Mbps	0.944

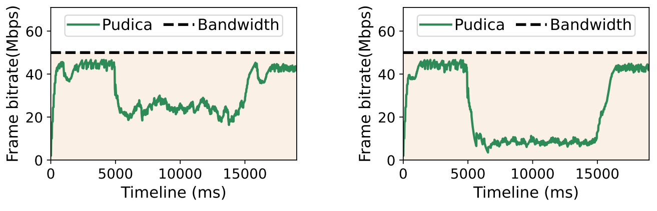
Table 4: Microbenchmarks: the performance of different Pudica variants when simultaneously launching three flows on the same link.

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >100/200ms**	**Avg. bitrate**
Vanilla Pudica	20.6ms	27.3/33.6ms	0.17%/0.009%	46.21Mbps
Pudica w/o temporary bitrate fallback	20.9ms	28.4/38.3ms	0.29%/0.031%	46.19Mbps
Pudica w/o active queue draining	23.7ms	36.0/60.5ms	1.40%/0.193%	47.16Mbps

Table 5: Microbenchmarks: performances of multiple Pudica variants to explain the contribution of our short-term BUR-driven control schemes.

Algo.	**Avg. delay**	**95%/99% -tile delay**	**Stall rate >100ms**	**Avg. bitrate**
Pudica	18ms	21.3/28.5ms	0.061%	47.86Mbps
α=0.8	17.9ms	21.3/29.1ms	0.075%	47.55Mbps
α=0.9	18.8ms	21.9/28.0ms	0.049%	48.28Mbps
γ_{MI}=0.20	18.5ms	22.1/30.7ms	0.084%	47.26Mbps
γ_{MI}=0.25	17.8ms	21.4/28.1ms	0.08%	48.14Mbps
ζ=10%	18.5ms	22.4/30.4ms	0.066%	47.99Mbps
ζ=20%	18.9ms	21.5/27.9ms	0.054%	47.42Mbps
T_{wd}=150	18.5ms	22.0/30.6ms	0.058%	47.96Mbps
T_{wd}=250	18.4ms	21.6/28.6ms	0.066%	47.82Mbps

Table 6: Sensitivity analysis for algorithm parameters.



(a) Buffer size: 10,000 bytes. (b) Buffer size: 50,000 bytes.

Figure 18: Pudica performance in the presence of a Cubic flow, with different buffer size settings of the bottleneck queue.

10s, respectively. We can see that Pudica exhibits a certain level of competitiveness when the buffer size of the bottleneck queue is small. However, when the buffer size increases, Pudica fails to compete effectively against Cubic.

I Difference Between Our BUR Estimation and Traditional Packet-Train Methods

Pudica leverages all packets within a frame as a short burst, which can be regarded as a packet train, to probe the link condition. The difference between our method and traditional packet-train network estimation approaches lies in what exactly to estimate and how to probe.

Traditional packet-train methods [43, 44] utilize the dispersion of a packet train, i.e., the gap in arrival time between the first and last packets, to estimate the available bandwidth or capacity. For a detailed definition of packet train and its estimation method, see [43]. In a single-flow environment, packet-train dispersion indicates the bottleneck link capacity, which also represents the available bandwidth. However, for a multi-flow scenario, the estimated value by these methods is actually neither available bandwidth nor capacity [45].

By contrast, Pudica leverages the queuing delay of a packet train to estimate the link utilization (i.e., BUR) during a frame interval. As per [43], a link is either transmitting at full capacity or idle at any given moment, meaning its instantaneous utilization is either zero or one. Therefore, BUR is defined as the time-averaged instantaneous utilization over a specific interval. In our case, this interval is the frame sending interval (i.e., L), typically 16.67 ms for a 60 frame rate. Then, the BUR of the time period $(t, t+L)$ can be expressed as:

$$R(t,t+L) = \frac{1}{L}\int_{t}^{t+L} r(x)dx \qquad (16)$$

where $r(x)$ is the instantaneous link utilization at time x.

Traditional packet-train probing methods typically implement a fixed *pace multiplier* (as defined in §2.1). These methods can only adjust the *pacing rate* by resorting to bitrate adjustments. In contrast, Pudica introduces an adaptive pace multiplier algorithm that enables appropriate queuing even when the bitrate remains unchanged. Another difference is that Pudica proposes a supplementary probing technique via a small number of probe packets beyond application data. This method enables Pudica to effectively probe the link condition even during the agnostic period.

Note that in Pudica, the BUR probing and estimation method introduced in §4.1 is only used when the link utilization is below one. When the link utilization exceeds one, the agnostic period is eliminated, and we rely on the packet receiving rate to estimate the available bandwidth (see §4.3), which is similar to traditional packet-train techniques.