Confucius: Adapting Home Routers to Congestion Control's Reactions for Consistent Low Latency

Zili Meng[1], Nirav Atre[2], Bochun Zhang[1], Mingwei Xu[3], Justine Sherry[4], Maria Apostolaki[5]
[1]*Hong Kong University of Science and Technology* [2]*Stanford University*
[3]*Tsinghua University* [4]*Carnegie Mellon University* [5]*Princeton University*

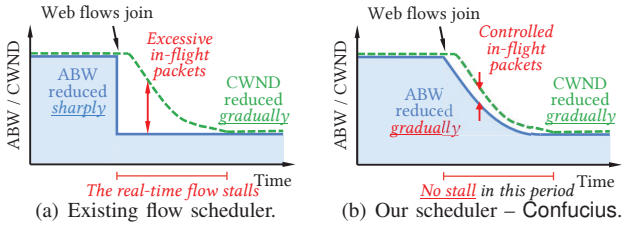
***Abstract*—Emerging high-quality real-time applications require consistently low latency, which is often disrupted by latency spikes. We identify the reason as the mismatch between the *abrupt* bandwidth reallocation on routers and *gradual* sending rate reaction of congestion control. For example, when a burst of new flows arrives, queue schedulers such as fair queueing immediately reallocate the bandwidth for existing and new flows. However, the flow's sending rate, determined by the congestion control algorithm (CCA), needs several RTTs to converge to the new available bandwidth, during which severe stalls occur. This has been increasingly critical with the demand on consistent low latency. In this paper, we present Confucius, a practical queue management scheme that reallocate the bandwidth for flows following CCA's reaction. Confucius slows down bandwidth adjustment to match the reaction of congestion control, so that the end host can reduce the sending rate without overshooting the network. Confucius is designed for offering real-time flows with consistently low latency regardless of uncertain competition. Experiments show that Confucius reduces the stall duration by more than 50% against existing practical schemes, while competing flows also fairly enjoy on-par performance.**

Available at: https://github.com/hkust-spark/confucius-qdisc

I. Introduction

High-quality real-time communications (RTC), including a range of applications from cloud gaming to VR/AR streaming, are becoming the dominant traffic on the Internet. These applications require low and consistent latency to maximize the user experience [1, 2] and also high throughput (§ II-A) due to their demand for high quality. Significant research has been dedicated to ensuring a satisfactory user experience through minimizing and stabilizing the end-to-end latency. Indeed, congestion control algorithms (CCAs) reduce the queueing delay [3, 4]; forward error correction (FEC) improves the loss recovery [5, 6]; multi-path transport mitigates fluctuation in wireless settings [7, 8]; while co-design with the video codec [9–11] and routers [1, 12] controls the delay in these components. Unfortunately, these works focus on *how to passively respond to network fluctuations after it happens*, instead of *how to actively minimize the network fluctuations*. As a result, latency fluctuations still routinely occur, causing deterioration of the performance of the real-time flow [13].

In this paper, we show that one root cause of the drastic network fluctuation is the unpredictable, transient flow competition on the last-mile routers (§ II). Our measurement shows that transient flow competition can be extremely drastic. For instance, loading a single Web page creates 9 concurrent flows on average since one page contains elements from many domains (Fig. 5(b)). Even these new flows might in the slow start phase, the initial congestion window (10+ packets) is already too much to the shallow buffer that low-latency CCAs try to maintain. This will drastically grab the available bandwidth from the real-time flows in a short period and cause stalls in multiple practical settings, e.g., home routers (§ II-B).



(a) Existing flow scheduler. (b) Our scheduler – Confucius.

Fig. 1: When new competing flows join, the available bandwidth (ABW) of the real-time flow will be immediately reduced, but the CCA takes multiple RTTs to converge.

The root cause for stalls here is the *mismatch between the scheduler's bandwidth reallocation on routers and CCA's reaction on endpoints*. If loading one website (9 flows) with an existing real-time flow, the available bandwidth of the real-time flow will *immediately* be reduced to 1/10 of what it was, as shown in the solid blue line in Fig. 1(a). CCAs reduce the congestion window or sending rate after end hosts observe latency increases or packet loss, but it is already too late. As shown in the green dashed line in Fig. 1(a), it takes several RTTs for CCAs to converge to the new available bandwidth, while the *excessive packets* during this period (marked in red) will have already resulted in a bufferbloat and led to a stall. For existing solutions, differentiated service (DiffServ) [14] (including L4S [15]) is not incentive-compatible in practice. Notification-based solutions, e.g., active queue management (AQM) [1, 16] and explicit congestion notification (ECN) [17], are reactive so the stall will still happen (§ II-D).

To this end, we designed Confucius[1], where the reduction of the available bandwidth of the existing flow can follow the reaction of the congestion control, as shown in Fig. 1(b). Instead of abruptly changing bandwidth allocations when a burst of new flows arrive, Confucius *smoothly* adjusts the bandwidth allocation to provide existing flows a few RTTs to react and converge to the change in network conditions. In this case, the excessively sent packets will be under control and

[1]Confucius' (the philosopher) educational philosophy is gradually teaching students following their reactions. In this paper, we schedule flows following their reactions as well.

the latency for the real-time flow can be maintained regardless of competing flows at the bottleneck.

Confucius has three goals of consistency, fairness and incentive-compatibility (§ III): First, Confucius needs to provide latency consistency to real-time flows independently of the number, rate, or CCA of the competing flows. Confucius achieves this by reallocating the bandwidth in a careful, exponential pattern, which comes with a theoretical guarantee for latency fluctuations experienced by real-time flows. Second, Confucius should eventually be fair – the performance of new flows should not be sacrificed too much either. Confucius quickly converges towards the fair allocation within a few RTTs with another theoretical bound. Finally, Confucius should be practical without relying on labels from end hosts so that router vendors have incentive to deploy. Therefore, Confucius does not perform any application grouping but directly builds upon per-flow scheduling, which is the default scheduler in most home routers.

We implement Confucius with both NS-3 simulator and kernel modules for Linux-based routers. Note that Confucius is designed for last-mile home routers (Fig. 2) where the available bandwidth fluctuates a lot due to competition [1], and home routers are mainly Linux-based[2]. With both trace-driven simulations and testbed experiments over live websites, we show that compared to FqCoDel (Linux's default) [19], Confucius reduces the stall duration of the real-time flow by 60%-69% and the loading time of Web pages by 39%-48% for top 1000 websites at the same time. Compared to other practical baselines, Confucius can still effectively reduce the stall duration of the real-time flows by at least 21% (§ V-B) with negligible computation overhead (§ V-D). In the meantime, long-lived, bulk transfers experience almost no degradation relative to fair queueing, and the impact on the flow completion time over short Web flows is limited to at most 10% even compared with the shortest job first (`SJF`, which strictly prioritizes the Web flows).

We will release all traces and codes of this paper. This work does not raise any ethical issues.

II. Motivation

A. The rise of real-time traffic

The home router has always been shared among multiple applications, but the proliferation of RTC applications has made it particularly challenging with two requirements: **(1) High bitrate.** New RTC applications such as cloud gaming stream at a bitrate of $>$ 30 Mbps [9, 20, 21]. In contrast, the worldwide average access bandwidth is only 50 Mbps [22] in November 2024, making the bandwidth still a scarce resource. **(2) Consistent low latency.** Real-time applications require not just low latency but *consistently* low latency while sending at high throughputs. For example, if the latency goes beyond 190 ms, cloud gaming users will immediately notice and might lose the game, incurring significant revenue loss [9].

[2]A measurement shows that 91% of home routers are Linux-based [18], where the default queue scheduler is FqCoDel [19].

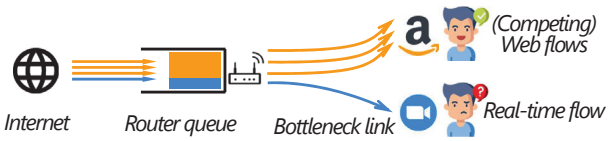


Fig. 2: The scenario where the real-time flow is affected by competing flows. When Web flows join the competition with the real-time flow, the available bandwidth of the real-time flow will be immediately reduced.

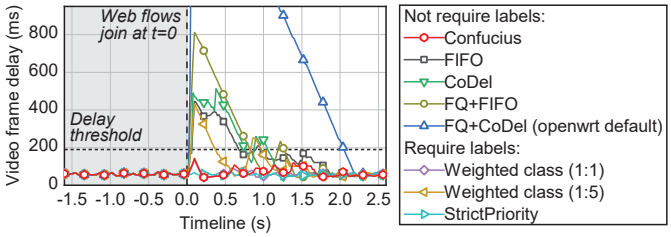


Fig. 3: An existing real-time flow competes with flows of loading the homepage of amazon.com. The real-time flow always experiences transient stalls during the competition unless flows are pre-labeled and differentiated by the router.

Setting & Scope: This work focuses on end-user access routers, where the congestion and latency fluctuation are dominant in end-to-end transport [1, 23, 24]. We acknowledge that one limitation of Confucius is that it is mainly designed for the scenario where real-time flows get congested at the home routers, and will not scale to core routers or delay-tolerant traffic. Nevertheless, the last-mile routers are still most likely to be the cause of jitter, irrespective of being wired [25] or wireless (5G, WiFi-6) [1, 26, 27]. Other congestions (*e.g.*, losses in the Internet core [28] or datacenters [29]) are usually not the root cause of the delay increase [1]. As most home routers are Linux-based [18, 30], they allow for flexible traffic management on software as the current practice [31].

B. Motivating example

To better illustrate the problem in existing approaches, we present an example in Fig. 2. Consider a user on a VR game, and others sharing the home router decides to load a Web page. Usually, the VR stream only has one RTC flow with heavy traffic. To showcase more baselines, we simulate the RTC flow's delay of each video frame using NS-3 and present the results in Fig. 3. The RTC flow [6] uses Copa, and the competing flows are captured from Chrome loading amazon.com using the default Linux kernel settings from v5.15.

Existing queue schedulers lead to drastic delay surges. Before t=0s, the sending rate of the real-time flow has converged, with the video frame delay fluctuating around 60ms. However, when the browser starts to load amazon.com, the delay for the real-time flow sharply increases. Using `FIFO`, the delay goes up to more than 400 ms, which is much higher than the stall threshold (190ms[3]) required by the application, and stays above the threshold for almost one second. When using per-flow fair queueing (`FQ`), the delay is even worse since it shifts more bandwidth away. The delay *always* surges regardless of

[3]This is the recommended network delay by ITU [32]. A more stringent threshold does not change the observation.

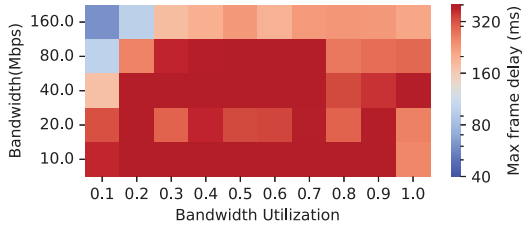


Fig. 4: We sweep the video bitrate and link capacity and measure the max frame delay when competing with live websites. The delay still surges even the capacity is high.

it using other CCAs like BBR or GCC (§ V-B).

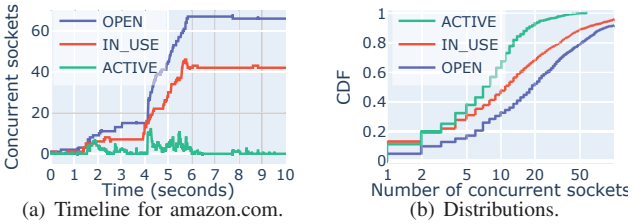
Is this a real and new problem? One might have never experienced the problem when using videoconferencing – this can be true. Videoconferencing flows, at around 2 Mbps [33] with low resolution and fixed headshots, are insufficient to overload a home network of 100 Mbps. However, the recent RTC flows consume significantly higher bandwidth (e.g., 30 Mbps for cloud gaming [9], 100+ Mbps for VR [20]).

Therefore, as long as the bitrate for video flows goes up, the problem will only be more severe. We repeat the experiment based on WebRTC by visiting the live website using Chrome (v120) and measure the maximum video frame delay in Fig. 3. We sweep the maximum bitrate of the real-time flow and the link capacity from 10 Mbps to 160 Mbps. The utilization is the video bitrate over bandwidth before visiting the website. We present the maximum delay suffered by the video frames in the heatmap in Fig. 4. As we can see, even with the increase of network bandwidth, as long as the bitrate demand from the real-time flow is still high, it will still suffer from hundreds of milliseconds of delay, and the highest (0.7, 40 Mbps) is more than one second. For example, when the network bandwidth is 160 Mbps, a 48 Mbps video flow (utilization 0.3) will still suffer from a maximum delay of 200+ ms, let alone the high bitrate of VR flows of 850 Mbps [21]. With the further increase of the bitrate for RTC flows, the problem will still persist.

C. Root cause analysis

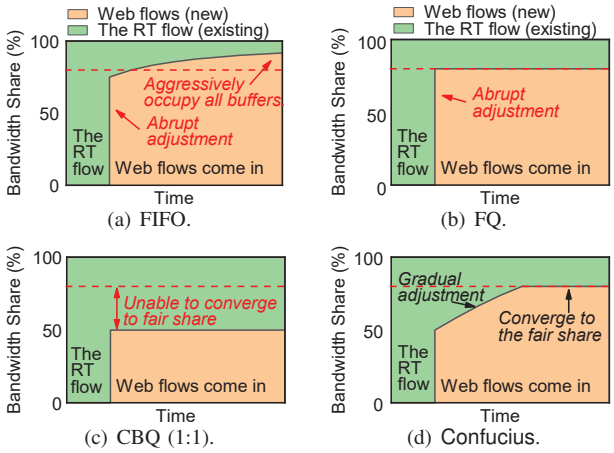
We argue that the delay spike is caused by (i) the burst of flows and packets from the competing Web page; (ii) the abrupt reallocation of the available bandwidth by queue management; and (iii) the gradual reaction from the congestion control. While these three fundamental design choices have long been considered integral to end-to-end efficiency, their concurrent exploitation creates a cascading effect, uncovered for the first time in this work. Next, we will elaborate on how these common factors result in performance degradation.

The source of the burst: One Web page always concurrently initiates multiple flows. To understand the spike in Fig. 3, we measure the flows launched by amazon.com over time. Concretely, we measure the number of sockets that are `OPEN` and `IN_USE` marked by NetLog from Chrome using `selenium`. We also measure the number of active flows every 10 ms that receive bytes through packet captures with `browsermobproxy` (`ACTIVE`). As shown in Fig. 5(a), loading only the homepage generates 68 flows in total, where



(a) Timeline for amazon.com. (b) Distributions.

Fig. 5: Number of *concurrent* flows recorded by `NetLog`. `OPEN` and `IN_USE` are socket states marked by Chrome, and `ACTIVE` means that the flow is receiving bytes in the last 10 ms.



(a) FIFO. (b) FQ. (c) CBQ (1:1). (d) Confucius.

Fig. 6: Illustration of how bandwidth shares change over time with incoming Web flows and the existing real-time (RT) flow for different schedulers. The dashed red line marks the fair share.

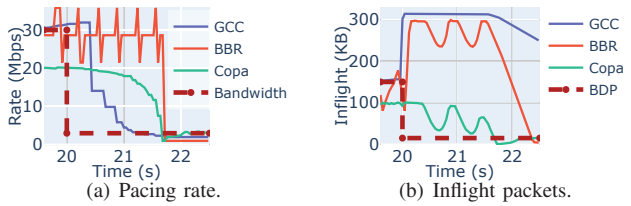
up to 12 flows run simultaneously. This is due to the Web design of hosting different objects (*e.g.,* images, ads) in various domains. Note that this is not due to the parallel connections in HTTP/1.1 – more than half flows go to different unique IPs.

Launching multiple flows to load one page is common across different websites. We measure the Alexa Top-1000 websites[4] and presented the distribution in Fig. 5(b). We find that the median number of concurrent `ACTIVE` flows is 8 while the 90th percentile is 19. The highest one in the Top 200, dailymail.co.uk, has 50 active flows at the same time.

Note that even new flows are in the slow start, they still have sufficient packets to compete with the existing flow in the bottleneck queue. The initial congestion window in Linux is already 10 packets, let alone that many CDNs increase it to improve the performance [34]. Meanwhile, low-latency CCAs will drain the bottleneck buffer as empty as possible [3, 4, 35] to below 10 packets.

The cause of the delay spike: Queue schedulers sharply reallocate service rates. We illustrate the bandwidth share of different queue management schemes in Fig. 6. For `FIFO` (Fig. 6(a)), the available bandwidth for the real-time flow will be drastically reduced since they share the same queue. `FQ` (Fig. 6(b)) makes matters worse – in our example, 12 new flows joining the `FQ` router directly reduces the available bandwidth of the real-time flow to 1/13.

[4]Although the Alexa Top website list has been deprecated, we still use this list since it is the most well-known list for top websites.



(a) Pacing rate. (b) Inflight packets.

Fig. 7: When the link capacity is reduced from 30 Mbps to 3 Mbps at t=20s, three CCAs respond differently. The CCA takes multiple RTTs to react and converge to the new link capacity.

The reason of delay spikes is that the CCA needs to *gradually* react and match its sending rate to the new available bandwidth, which takes several RTTs (dashed green line in Fig. 1). During this time, the excessive in-flight packets will result in high latency for the real-time flow. AQM notifies the sender about the network conditions by ECN on packets but cannot prevent stalls either since it is still after the queue length increase, as shown in Fig. 3. We observe similar limitations in other AQMs (§ V-B).

The fact hard to change: CCA takes a long time to converge. The issue is that the available bandwidth for the real-time flow drops immediately, but the end-to-end CCA cannot immediately fit the new available bandwidth. We test three low-latency CCAs (GCC [3], Copa [4], and BBRv1 [35]) and see how they react to bandwidth changes in Fig. 7(a) (settings in § V-A). We reduce the bottleneck link capacity from 30 Mbps to 3 Mbps and measure the sending rates of three CCAs. The fastest reaction, GCC, still takes 0.4 seconds to react and 1.2 seconds to converge to the new available bandwidth. We also measure the size of inflight packets, which ideally should be the bandwidth-delay product (BDP) to minimize the buffer in the network in Fig. 7(b). BDP decreases even after the sending rate converges.

Simply making the CCA more agile will lead to the increase of delay [3, 36]. Helping the CCA to quickly converge using in-network information [37] needs both end hosts and routers to collaboratively deploy, which poses significant barriers to deployment on the Internet [1]. Moreover, during the convergence of the CCA, the excessive in-flight packets also inflate the RTT, slowing down the update as well. Putting all factors together, existing solutions cascadingly result in the delay spike of the real-time flow.

D. Related works and limitations

Although previously not designed for the problem of abrupt change in § II-C, one might think of several related works. However, None of these solutions work well.

A better access link? Upgrading the bandwidth of the access link can alleviate the issue but cannot solve it. As we present in § II-B, even if the access link is 1 Gbps, as long as the bitrate demand of real-time video streams increases, the problem still exists. Meanwhile, upgrading the infrastructure is never easy – 63% of users in the States are still using coaxial cables in 2023 despite the fibre optics has already been the status quo for decades, and only 1/3 households have an "advertised" access speed of 1 Gbps or higher [38].

DiffServ [14] schedules them differently on the router using StrictPriority or weighted class as shown in Fig. 3. This also includes L4S [15] which we will later evaluate in § V-B. Yet, `StrictPriority` towards the real-time flow will drastically harm the performance of competing Web flows (§ V-B). Allocating bandwidth weights for different classes needs accurate estimation of the fluctuating bandwidth demands, where inaccurate estimation easily leads to unfairness or latency spikes (Figs. 3 and 6(c)).

Different identifiers. Scheduling by IP pairs instead of 5-tuples can aggregate multiple flows for the same pair (e.g., parallel connections in HTTP 1.1), but in our case, Web flows have completely different IPs. Scheduling by the application will require client information, which is not incentive-compatible and can be compromised. Scheduling by the destination IP or per-user can help with the competition between different users but is again ineffective when flows come from the same user.

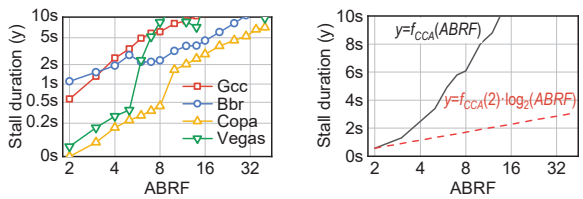
Other reactive mechanisms. Zhuge [1] reduces the feedback loop between the router and the endpoint but does not accelerate CCA's convergence (§ II-C). Using the example in Fig. 1, Zhuge shortens the distance between the turning points of the blue and green lines, but the dominant contributor – the time for the green dashed line to converge to the blue line – persists. FEC [5, 6] is hardly helpful in our case since most of them have no loss at all. Multipath transport will switch to the new path [7, 8], but this also occurs *after* the sender observes drastic degradation. Bandwidth estimations from lower layers [39, 40] are not effective either since the link capacity does not change in the competition.

III. Confucius Design Requirements

R1: The performance of the real-time flow should be constant to arbitrary competing flows. Confucius stands out among queue management algorithms in that it theoretically guarantees worst-case performance, no matter what CCAs and competing flows are. This will, in consequence, fundamentally address the root cause of latency fluctuation induced by unpredictable competing traffic. We theoretically calculate performance bounds for a few classes of applications in § IV-D. We demonstrate that with Confucius, real-time flows have a near-constant bound of latency degradation (around 250 ms in § V-C), no matter how large and how many competing flows join the bottleneck.

R2: Latency consistency should not come at the cost of long-term fairness. Confucius should still follow per-flow fairness in the long run. To do so, Confucius moves rates towards a fair allocation quickly and pushes the blue solid line in Fig. 1(b) to match the green dashed line. Confucius adjusts flow service rates exponentially as shown in Fig. 6(d), converging to the fair share in several RTTs. Our experiments (§ V-C) and theoretical analysis (§ IV-D) show that the degradation is not inflated with fairness maintained.

R3: The identification of flows should not rely on end hosts. To make Confucius incentive-compatible and deployable in



(a) Measurements (b) Illustration

Fig. 8: (a) Stall duration increases with the available-bandwidth-reduction factor (ABRF). (b) An illustration of how smoothly reducing available bandwidth helps reduce stall duration. Note that (a) is a log-log plot but (b) is a log-lin plot.

practice, we aim to identify the flows of interest *at the router itself*, without relying on end hosts. The performance improvements should be directly observed by the router vendor without going through endless coordination between end-host content providers and router vendors in IETF.

IV. Age-aware Flow Weights Adjustment

We explain the benefits of exponential bandwidth reallocation (§ IV-A) and dive into Confucius' weight adjustment mechanism (§ IV-B and IV-C). We further explain the theoretical features that Confucius can provide (§ IV-D), and discuss for better understandings (§ IV-E).

A. Exponential Bandwidth Re-allocation

We quantitatively demonstrate the advantage of *smoothly* controlling the real-time flow's bandwidth allocation compared to directly cutting its available bandwidth. We measured the stall duration y for the real-time flow in the scenario of a sudden reduction of available bandwidth for four low-latency CCAs (§ IV) with the same setting in § II-B. We plot y as a function of the Available Bandwidth Reduction Factor (ABRF, the factor we reduce the available bandwidth) for different CCAs in Fig. 8(a). CCAs respond poorly to sudden, large reductions in bandwidth – for instance, reducing Copa's available bandwidth to 1/10 of its initial value (i.e., $ABRF = 10$) results in a $y > 2$ *seconds* stall (ref. Fig. 7(a)).

Confucius *smoothly* reduces the available bandwidth in an exponential way. For instance, to reduce the bandwidth by 16×, we can reduce the bandwidth four times, each by half. Fig. 8(b) demonstrates that, ideally, compared to the super-linear stall duration (solid line copied from Fig. 8(a)), exponentially reducing the sending rate will only increase the stall duration *logarithmically* with the ABRF (modulated by $f_{CCA}(2)$, a small constant). We will elaborate on how to realize this in the next subsection.

B. Weight Adjustment Mechanism

For each flow f, Confucius computes a weight, w_f. The bandwidth share for that flow will be its weight over the sum of all weights per the scheduling of Deficit Weighted Round Robin (DWRR). We present the weight adjustment algorithm in Alg. 1. We denote the set of new flows and all flows as $\mathcal{F}_{new}$ and $\mathcal{F}_{all}$, where $\mathcal{F}_{new} \subseteq \mathcal{F}_{all}$. We will give the formal definition of new flows in § IV-D. Here we first focus on how the weights are adjusted in the runtime:

Algorithm 1: Reweight: run every $1/\lambda$ ms.

```
1  Reweight ()
2      foreach f ∈ F_new do
3          w_f ← 2 w_f // The weight doubles every 1/λ ms
4      ratio = (Σ_{i∈F_new} w_i) / (Σ_{i∈(F_all − F_new)} w_i)
5      if ratio < 1 then // New flows are the minority
6          foreach f ∈ F_new do
7              w_f ← w_f / ratio
8      foreach f ∈ F_new do
9          if w_f ≥ 1 then // The new flow is old enough
10             F_new ← F_new − {f}
11             w_f ← 1
```

Age-aware exponential adjustment (lines 2-3). For all new flows $f \in \mathcal{F}_{new}$, the weight w_f will be doubled every $1/\lambda$ milliseconds. This shows how Confucius *exponentially increases* the weights of new flows. λ controls the speed for the weight adjustment. We discuss how to determine λ in § IV-D.

The design also has a consideration of runtime overhead. We double the weight periodically every $1/\lambda$ ms by left-shifting the weight by one bit every $1/\lambda$ ms for the exponential weight updates to save runtime overhead in Linux kernel.

When there are more old flows than new flows (lines 4-7). Confucius addresses the issue where a burst of new flows impacts a few existing flows. However, there may be more existing flows than new flows (line 5). In this case, we allocate resources to new flows as long as the bandwidth reduction of existing flows is bounded by half and the fair share.

When flows are no longer new (lines 8-11). Confucius uses a flow weight threshold of 1 to 'age out' new flows from $\mathcal{F}_{new}$.

C. Arrival and completion of new flows

Algorithm 2: When a flow f arrives.

```
1  NewFlowArrival (f)
2      foreach i ∈ F_new do // Scale down weights
3          w_i ← (|F_new| / (|F_new|+1)) w_i
4      w_f ← 1 / (|F_new|+1) // Initialize the weight for f
5      F_new ← F_new ∪ {f} // Add f to F_new
```

When a flow f just arrives at the queue, Alg. 2 will adjust the weights of the flows that are already in $\mathcal{F}_{new}$ (lines 2-3), rebalance the weights for flow f (line 4), and add f to $\mathcal{F}_{new}$ (line 5). Alg. 2 has two good features:

The sum of weights of new flows is upper-bounded. Rebalancing the weights of the new flows $\mathcal{F}_{new}$ to the just arrived flow f (lines 2-3) can make sure the existing flows will not be affected with the arrival of flow f.

The ratio between weights of flows in $\mathcal{F}_{new}$ maintains. During the rebalancing, the flows arriving earlier will still proportionally have larger weights. This will help to maintain the long-term fairness (**R2** in § III).

Similarly, as shown in Alg. 3, when one flow completes when it is still in $\mathcal{F}_{new}$, we will reallocate the bandwidth share to all flows in $\mathcal{F}_{new}$. This ensures the new flows can converge fast without affecting the existing flow. From the router's view,

Algorithm 3: when a new flow $f \in \mathcal{F}_{new}$ completes.

```
NewFlowCompletion (f)
    foreach i ∈ F_new do // Scale up weights
        w_i ← (|F_new| / (|F_new| − 1)) w_i
    F_new ← F_new − {f} // Remove flow f
```

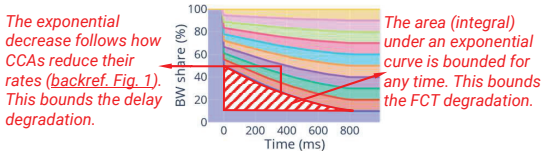


Fig. 9: Illustration of theoretical benefits of Confucius for both the real-time flow and competing flows. This is the example analyzed before in § II-B. The bandwidth share of the existing flow (in blue) will decrease gradually and converge to the fair share.

we consider the flow as completed when it does not show up for a period of time. We periodically check every $1/\lambda$ (the response time of existing CCAs) together with Alg. 1.

Discussions: App-limited flows. Some Web pages will reuse the same TCP connection to load different objects from the same IP, resulting in app-limit behaviors. If the gap between two bursts of traffic is within the timeout (§ IV-C), the flow will not be re-identified as new flow and the weight will keep increasing. Otherwise, we have to consider the flow as a new flow since the existing flows have already converged to new bandwidth shares. In our evaluation, many Web flows are app-limited and Confucius still has satisfactory improvements.

D. Theoretical Analysis

The performance guarantees of Confucius over Web flows and the real-time flows come from the exponential reweight mechanism. We still follow the same example in § II-B. At $t = 0$, N new flows, each with size B bytes, join the same bottleneck link and compete with the existing flow. We show that Confucius guarantees bounded stall for the existing real-time flow while yielding FCTs for Web flows within a constant additive factor of what FQ provides.

In this example, Confucius sets the weights for new flows following $2^{\lambda t}$, starting from $1/N$ (Alg. 2). So the available bandwidth for the existing flow (in blue) will be $\frac{1}{1+2^{\lambda t}}$ of the link capacity. It has two good features as shown in Fig. 9:

- The area under this curve is the additional amount of real-time traffic in Confucius compared with FQ, which also upper-bounds the degradation over the competing Web flows. Due to the convergence of the integral of this function ($\int_0^\infty \frac{1}{1+2^{\lambda t}} \mathrm{d}t = 1$), the degradation to the competing flows is also bounded, regardless of the number N and the size B of competing flows.
- The decrease of the available bandwidth of the existing flow is almost exponential ($\frac{1}{2} \cdot 2^{-\lambda t} \leq \frac{1}{1+2^{\lambda t}} \leq 2^{-\lambda t}$), which matches the decreasing curve of many CCAs (ref. Figs. 1 and 7). The bufferbloat is the area between two curves in Fig. 1(b), and the stall duration will also be bounded.

We analyze mathematically and summarize the results using Taylor expressions in Tab. I. Confucius ensures that the stall duration (q_P^{max}) for real-time flows is upper bounded,

Policy P	q_P^{max}	$T_P - T_{FQ}$
Confucius	$\approx 6q_0 + 15\tau + \frac{8\lambda}{k} + \frac{(10q_0+15\tau)\lambda^2}{k}$	$\approx \frac{\log_2 e}{\lambda}$
FQ	$\approx \boldsymbol{N}\left(\frac{2}{3}\sqrt{\frac{2}{k}} + q_0 + \tau\right)$	0
FIFO	$\approx \left(\frac{\boldsymbol{N}B_0}{q_0 C} + 1\right)\left(\frac{2}{3}\sqrt{\frac{2}{k}} + q_0 + \tau\right)$	$\lessapprox 0$
CBQ	$\approx \frac{2}{3}\sqrt{\frac{2}{k}} + q_0 + \tau$	$\approx \frac{(\boldsymbol{N}-1)\boldsymbol{B}}{C}$

TABLE I: Approximations for different schedulers P on their maximum queueing delay (q_P^{max}) and FCT degradation against FQ ($T_P - T_{FQ}$). Confucius has a bounded performance degradation for all flows. In the competition, existing schedulers have either unbounded delay, or unbounded FCT degradation marked in red.

only depending on the CCA's latency sensitivity (q_0), the responsiveness (k), the feedback loop (τ), and Confucius' parameter (λ). Confucius can also ensure the FCT degradation for new flows ($T_P - T_{FQ}$) is bounded by an additive constant factor to the decay parameter (λ), which goes to negligible with the increase of the flow sizes.

For FQ and FIFO, the stall duration scales up with the number of new flows, N, and is therefore unbounded. For class-based queues (CBQ), since the pre-defined weights do not match the traffic ratio, CBQ converges unfairly, and the flow completion time (FCT) degradation for new flows becomes unbounded.

How to determine λ. Here we set λ=0.004 (ms^{-1}) to have a doubling interval of $1/\lambda$=250 ms, where the PLT degradation for typical websites of 10-20 flows is less than 100 ms, acceptable compared to the second-level PLT. Even where different CCAs have various and mixed response times (Fig. 7(a)), such a setting work for common CCAs (Fig. 17).

E. Limitations

We outline some limitations of Confucius here.

Applications using latency-sensitive CCAs. In this paper, we assume that real-time applications use latency-sensitive CCAs. This general holds since the operators of real-time applications will optimize towards latency. Otherwise, the application's CCA itself is still problematic.

Web flow characteristics for mobile applications. The measurement of Web flows in § II is loading Web pages from desktop browsers (Google Chrome). We do not conduct measurements over mobile Apps but the root cause contributing to the bursts still exists – one page contains diverse objects from different domains.

The bitrate of real-time flows. Whether adjusting the bandwidth allocation abruptly or smoothly, the bitrate of the existing flows will be impaired. This is unavoidable since RTC traffic always prioritizes latency over bitrate [9].

V. Evaluation

We evaluate the performance of Confucius with both Linux-based router implementation over live websites, and large-scale trace-driven simulations in different network conditions. We first present our experimental setup (§ V-A); then we evaluate Confucius by the following perspectives:

- Confucius reduces the stall duration of a real-time flow by 21% to 87% while maintaining comparable FCTs (§ V-B).

- Confucius is consistently performant with different sizes and numbers of Web flows (§ V-C).
- We conduct live experiments on popular websites and results show that Confucius reduces the stall duration by more than 60% with reasonable overhead (§ V-D).
- We further show that Confucius can outperform baselines when working with multiple real-time flows, bandwidth-probing CCAs, and different bottlenecks (§ V-E).

A. Experiment Setup

Linux kernel setup. We build a real-world testbed to evaluate the performance of Confucius For the RTC flow, we test with two settings: (i) a WebRTC application based on M119 branch with GCC; and (ii) a TCP socket application running over Copa (used by Meta Live [41]). We set up three virtual machines (running Ubuntu 22.04) over a Dell PowerEdge R760 server with two Intel Xeon Gold 6430 CPUs, serving as the server, router, and client for the real-time flow.

At the client, we also deploy a selenium-driven Chrome browser (v120) to automatically load the websites. We add a delay for the RTC flow only, and set the bandwidth capacity on the router to be 20 Mbps following settings in [1, 42].

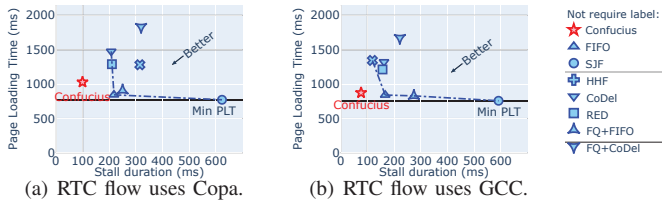
We implement Confucius into the Linux queue discipline. We patch the `sch_confucius.c` to the Linux kernel v5.15, and `q_confucius.c` to `iproute2`. They together have 1.2k lines of codes (LoCs). The queue discipline is compiled into a kernel module and inserted to the Linux-based router. The tests are conducted in September 2024 (§ V-D).

Simulation setup. We also implement Confucius into the queue discipline in `ns-3.34` in § V-B and V-C. We use the example in Fig. 2 and set the capacity of the bottleneck link based on the bandwidth datasets from [1]. The dataset contains mixed cellular traces of 4G and 5G, where the average bandwidth ranging from 22 Mbps to 375 Mbps. We adopt the RTC library in `ns-3` from [1, 6] as well. We evaluate the real-time flow with two delay-sensitive CCAs, Copa [4] and GCC [3]. The Web flows use Cubic [43] but note that only in simulation do the Web flows use the CCA set by us. In the live experiments, the Web flows have diverse CCAs configured by the website developers, including BBR and others [44].

The workloads of the competing Web flows are collected from the Alexa Top 1000 websites, also using the same Chromium and selenium driver. We use browsermobproxy to record the HTTP requests and packet captures and replay them in ns-3. Simulation results are in § V-B and V-C.

Baselines. We compare Confucius with multiple schedulers that do not require labels listed below. We use the default parameters in the Linux kernel 5.15.0 and `ns-3.34`.

(1) `FIFO` and (2) `FQ+CoDel` [19], the default qdiscs in Linux (before and after systemd v217 [45]).
(3) `FQ+FIFO` is the fair queueing without the AQM.
(4) `CoDel` [16] and (5) `RED` [46] will drop packets before the queue overflows to notify the sender.
(6) `SJF` (shortest job first) prioritizes short flows over long flows, which is exactly opposite to what Confucius tries to do. We take the implementation from PIAS [29].



(a) RTC flow uses Copa. (b) RTC flow uses GCC.

Fig. 10: The trade-off between the RTC flow (stall duration) and Web flows (PLT) on bandwidth traces `C1`. The dashed line denotes the Pareto front of baselines that do not require labels.

(7) `HHF` [47] (heavy-hitter filter) differentiates between small flows and heavy-hitters and schedules separately.

Metrics. We focus on the following metrics in experiments.

- *Stall duration* for video frames is the duration for which the delay of the video frame is greater than 190 ms. This reflects users' experiences on video stalls [1, 32]. We use this metric to evaluate how the RT flow is affected.
- *Page Load Time (PLT)* is the time till the last HTTP request in a web page is completed. We use this metric to evaluate the performance of web traffic. PLT degradation refers to the increase of delay compared to `FQ`.

B. Confucius under a realistic workload

Simulation scenario. We have a long-running real-time flow from the RTC module in ns-3. We then randomly select a website and replay the traces we collected to compete with the real-time flow. We set the interval of loading two websites to 53 seconds, the average Web page viewing time [48]. In each run, we measure the duration where the frame delay of the video flow is larger than 190 ms (stall). We also measure the Web PLT for websites. We present the average PLTs and stall durations in Fig. 10, and dive into distributions later.

Confucius strikes a balance between video and web performance that is consistent across CCAs. In Fig. 10(a), we observe that schedulers not relying on labels from the end host (marked in blue) suffer from long video stalls. For example, when the real-time flow adopts Copa, using `FQ-CoDel` or `FIFO`, the real-time flow experiences a stall of 200 ms to 300 ms on average when loading different websites. In contrast, Confucius can reduce the stall duration by more than a half to less than 100 ms. Most importantly, Confucius does not incur too much penalty on the PLT of Web pages, and pushes the Pareto front of the schedulers not requiring labels (the dashed blue line) forward. Confucius reduces the PLT compared to `CoDel`, `RED`, `FQ+CoDel`, and `HHF` since Confucius gently adjusts the bandwidth share for the Web flows. Even compared to FQ+FIFO, Confucius only increases the average PLT by *up to 8%* in three subfigures, which is much smaller than the improvement on the stall duration.

Confucius has comparable performance to mechanisms requiring active labels from end hosts. We also evaluate four baselines that require labels from end hosts:

(1) `DualQ+Prague`. `DualQ` [49] is a recently proposed scheduler in L4S [15] that protects latency-sensitive flows with labels, using DSCP bits to identify the traffic and notify the sender. It works best with TCP Prague in the

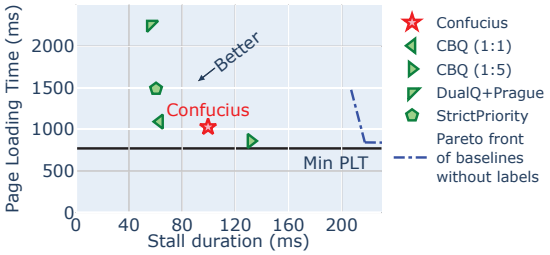
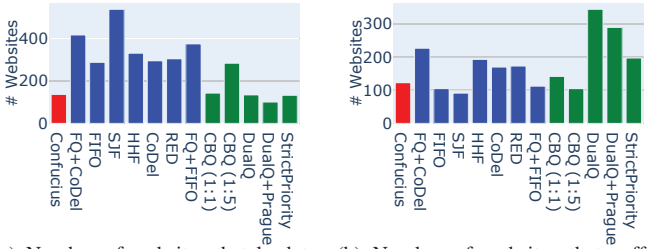


Fig. 11: The trade-off between the real-time (RT) flow (stall duration) and Web flows (PLT) on bandwidth traces `C1` using Copa compared with baselines that require labels from end hosts. The dashed line denotes the Pareto front of baselines that do not require labels from end hosts.



(a) Number of websites that lead to non-zero stalls for the RT flow.

(b) Number of websites that suffer from PLT of longer than 2 seconds.

Fig. 12: The number of runs, out of 1000 websites, that lead to the degradation of the real-time and Web flows. The lower the better. Fig. 12(a) cuts Fig. 13(a) at 190 ms.

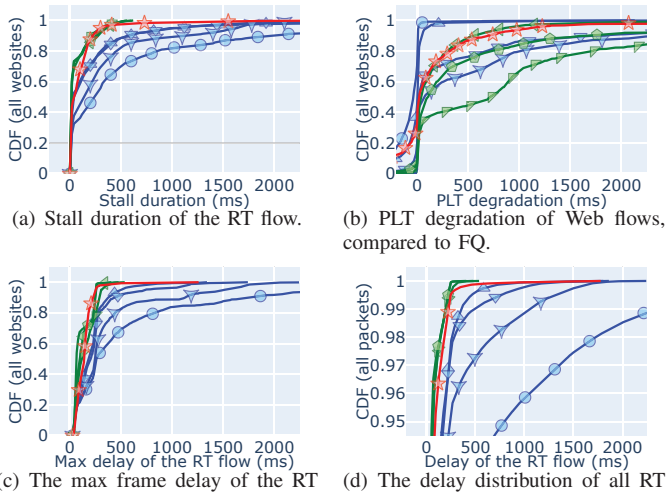
L4S framework. We implement from [50].

(2) `CBQ (1:1)` and (3) `CBQ (1:5)` are the weighted class-based queues, which put flows into different classes based on application labels. We set the weights for two classes (RT:Web) to 1:1 and 1:5 respectively.

(4) `StrictPriority` strictly prioritizes traffic from real-time flows if they are labeled accordingly.

Schedulers requiring labels (marked in green) protect prelabeled RT flows, but considerably degrade the PLT for the Web traffic. `DualQ+Prague` improves `DualQ` since the CCA on the end-host can react more effectively, but still incur considerable penalty on Web flows. Note that even when using `StrictPriority`, the real-time flow still suffers from 60 ms degradation on average due to bandwidth fluctuation. Confucius is almost on par with schemes relying on end-host labels in terms of the delay. Remember that it is unrealistic to assume that an end-host will correctly label all traffic (§ II-D).

Confucius protects the real-time flow when competing with traffic from 86% of the websites. We further break down the details for different websites in Fig. 10(a). In Fig. 12(a), we present the number of websites that do not affect the delay of the RTC flow (the stall shorter than 190 ms). The lower the number is, the better the performance of the scheduler is. When using Confucius, the real-time flow will only suffer from stall when competing with 136 out of 1000 websites. However, for all other baselines that do not require labels, the number ranges from 288 websites (`FIFO`) to 537 websites (`SJF`). Confucius reduces the number by 53%-75%. Even for baselines requiring labels, the number ranges from 101 to 293. Confucius can achieve comparable or better



(a) Stall duration of the RT flow.

(b) PLT degradation of Web flows, compared to FQ.

(c) The max frame delay of the RT flow when Web flows arrive.

(d) The delay distribution of all RT packets during the competition.

Fig. 13: The distributions in Fig. 10(a). The legend is the same as Fig. 10. We present a part of the baselines for simplicity.

protection to the real-time flow as those label-based solutions.

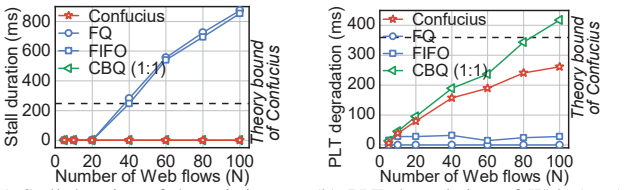
We also measure the number of websites suffering from a long PLT of longer than 2 seconds, which is the threshold for good user experience [51]. When using `FQ+CoDel`, 227 websites will suffer from long PLT, while Confucius can reduce this number by half to 127. Confucius's results is comparable to `FQ+FIFO`, demonstrating the fairness of Confucius for competing Web flows. For baselines requiring labels that behave well in Fig. 12(a), at least 198 websites suffer.

Confucius controls the delay and PLT following the theoretical analysis. Fig. 13(a) further presents the distribution of stall duration when the video flow encounters Web flows from different websites in the dataset. With `FQ+CoDel` or `FIFO`, the stall for the real-time flow will last for longer than 500 ms for 12% (`FIFO`)-18% (`FQ+CoDel`) websites, where the number for Confucius is 1%. In contrast, with Confucius, the real-time flow will not experience any stall when encountered with 95% of the websites, comparable to `CBQ`. Importantly, besides the PLT measured in Fig. 12(b), Confucius does not over-penalize web traffic – 60% of websites will not suffer from a penalty at all against `FQ`, as shown in Fig. 13(b), which mostly corroborates our previous theoretical analysis. We further present the distribution of maximum experienced delay for the real-time flow in Fig. 13(c). The fraction of having a maximum delay of >500 ms is 1% using Confucius, while for `FIFO` and `FQ+CoDel` are 5% and 18%. This further demonstrates that Confucius can control the latency fluctuation in not only the stall duration but also directly the raw delay. The dive into the network delay of all packets of the real-time flow in Fig. 13(d) corroborates this as well.

C. Confucius under workload changes

In this subsection, we test our theoretical analysis by investigating whether Confucius can provide consistent performance in controlled workloads.

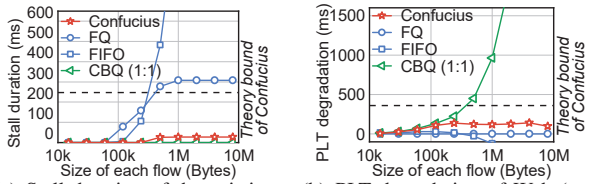
Confucius is bounded by theoretical thresholds, confirming our analysis. We vary the number of Web flows from 5



(a) Stall duration of the existing real-time flow.

(b) PLT degradation of Web (new) flows against FQ.

Fig. 14: Performance consistency in workloads with different *number* of Web flows, each flow with the size of 15KB.



(a) Stall duration of the existing real-time flow.

(b) PLT degradation of Web (new) flows against FQ.

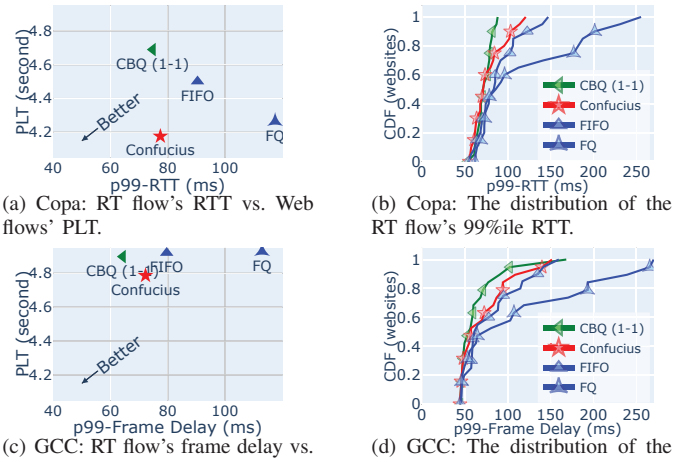
Fig. 15: Performance consistency in workloads with different *size* of Web flows, each experiment having 5 flows. The dashed line is the theoretical bounds from Tab. I.

to 100, each with the size of 15KB (medium flow size in our measurement), and summarize our results in Fig. 14(a). The stall duration for FQ and FIFO increases with the number of flows: when the number of Web flows goes to 60, the real-time flow experiences a stall for more than half a second. Confucius maintains zero stall in this setting, similar to CBQ. We further compare the experimental results with our previous theoretical analysis in § IV-D, as the dashed line in Fig. 14.

We change the size of Web flows as well. With the increase of the flow size, the competing flows are changing from short flows (e.g., Web) to long flows (e.g., FTP). Confucius is still able to achieve negligible stall for the real-time flow and bounded PLT degradation for Web flows at the same time.

D. Live Website Experiments

We stream the video frames at 60fps using two different applications – (i) socket with TCP Copa, and (ii) a simplified WebRTC implementation as in § V-A using GCC. We measure



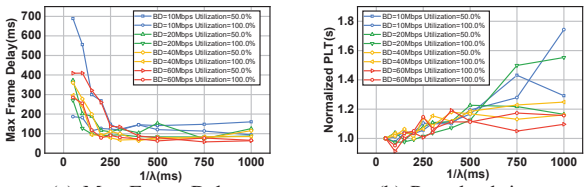
(a) Copa: RT flow's RTT vs. Web flows' PLT.

(b) Copa: The distribution of the RT flow's 99%ile RTT.

(c) GCC: RT flow's frame delay vs. Web flows' PLT.

(d) GCC: The distribution of the RT flow's 99%ile frame delay.

Fig. 16: Results over live websites in the real world.



(a) Max Frame Delay

(b) Page load time

Fig. 17: We sweep λ and measure the maximum frame delay and page load time of live websites.

the RTT from the socket statistics, and the delay for each video frame in the WebRTC example. We load the web page using Chrome from the top 20 websites every 53 seconds following § V-A. Since during each loading the content of the websites can differ, we repeat the experiments 5 times for each website and record all the results. Since WebRTC does not have per-packet acknowledgement, we directly measure the frame delay. We measure the 99th percentile delay for the real-time flow and the PLT of the website.

As shown in Fig. 16(a), Confucius reduces the 99th percentile RTT by 1/3 with Copa, consistent with our simulation in Fig. 10(a). This result is similar to our simulation in Fig. 10(a). Moreover, from our experiments, Confucius can also reduce the PLT by around 10% compared to CBQ because CBQ cannot converge to the allocation that is fair to the Web flows. Note that the axes in Fig. 16(a) do not start from zero for the clarity. The latency improvement with Confucius is consistent across all experiments, as illustrated in Fig. 16(b), where Confucius outperforms FQ and FIFO. Confucius also achieves a similar delay as CBQ for the majority of websites, and this improvement is also evident in the application's frame delay, as shown in Figs. 16(c) and 16(d).

Fig. 17 illustrates the impact of sweeping λ. As $1/\lambda$ increases, maximum frame delay decreases while page load time increases. The optimal value for $1/\lambda$ is 250 ms, balancing the performance of both RT-flow and Web flows.

E. Microbenchmarks

We further evaluate the performance of Confucius in a series of microbenchmarking settings. We show that the computational overhead of Confucius in Linux is close to the current default one, FqCoDel. We further find that Confucius will not have side effects on the fairness aspect with a JFI always higher than 0.95 with 1 to 40 competing flows.

VI. Conclusion

We propose Confucius, the first queue management scheme to protect the real-time flows in the flow competition in a practical way. Confucius achieves this by gradually adjusting the service rates of flows to match the reaction of congestion control. Doing so allows Confucius to mitigate latency spikes of real-time flows. Extensive evaluation shows that Confucius protects the real-time flows from stalls when competing with 86% websites, doubling over numerous baselines.

Acknowledgement. This work is supported by RGC Early Career Scheme (No. 26212525) and NSFC (No. 62221003). Mingwei Xu is the corresponding author.

REFERENCES

[1] Z. Meng, Y. Guo, C. Sun, B. Wang, J. Sherry, H. H. Liu, and M. Xu, "Achieving Consistent Low Latency for Wireless Real Time Communications with the Shortest Control Loop," in *Proc. ACM SIGCOMM*, 2022.

[2] X. Huang, J. Xu, H. Wang, H. Yu, S. D. Sathyanarayana, S. Shi, and Z. Meng, "Ace: Sending burstiness control for high-quality real-time communication," in *Proc. ACM SIGCOMM*, 2025.

[3] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, 2017.

[4] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX NSDI*, 2018.

[5] M. Rudow, F. Y. Yan, A. Kumar, G. Ananthanarayanan, M. Ellis, and K. Rashmi, "Tambur: Efficient loss recovery for videoconferencing via streaming codes," in *Proc. USENIX NSDI*, 2023, pp. 953–971.

[6] Z. Meng, X. Kong, J. Chen, B. Wang, M. Xu, R. Han, H. Liu, V. Arun, H. Hu, and X. Wei, "Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming," in *Proc. USENIX NSDI*, 2024.

[7] Y. Ni, Z. Zheng, X. Lin, F. Gao, X. Zeng, Y. Liu, T. Xu, H. Wang, Z. Zhang, S. Du *et al.*, "Cellfusion: Multipath vehicle-to-cloud video streaming with network coding in the wild," in *Proc. ACM SIGCOMM*, 2023, pp. 668–683.

[8] S. Dhawaskar Sathyanarayana, K. Lee, D. Grunwald, and S. Ha, "Converge: Qoe-driven multipath video conferencing over webrtc," in *Proc. ACM SIGCOMM*, 2023, pp. 637–653.

[9] Z. Meng, T. Wang, Y. Shen, B. Wang, M. Xu, R. Han, H. Liu, V. Arun, H. Hu, and X. Wei, "Enabling high quality real-time communications with adaptive frame-rate," in *Proc. USENIX NSDI*, 2023.

[10] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein, "Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol," in *Proc. USENIX NSDI*, 2018.

[11] Y. Cheng, Z. Zhang, H. Li, A. Arapin, Y. Zhang, Q. Zhang, Y. Liu, K. Du, X. Zhang, F. Y. Yan *et al.*, "Grace: Loss-resilient real-time video through neural codecs," in *Proc. USENIX NSDI*, 2024, pp. 509–531.

[12] W. Chen, L. Ma, and C.-C. Shen, "Congestion-aware mac layer adaptation to improve video teleconferencing over wi-fi," in *Proceedings of ACM Multimedia Systems Conference (MMSys)*, 2015.

[13] H. Chang, M. Varvello, F. Hao, and S. Mukherjee, "Can you see me now? a measurement study of zoom, webex, and meet," in *Proc. ACM IMC*, 2021, pp. 216–228.

[14] F. Baker, J. Babiarz, and K. H. Chan, "Configuration Guidelines for DiffServ Service Classes," IETF RFC 4594, 2006.

[15] B. Briscoe, K. D. Schepper, M. Bagnulo, and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture," RFC 9330, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9330

[16] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, 2012.

[17] A. M. Mandalari, A. Lutu, B. Briscoe, M. Bagnulo, and O. Alay, "Measuring ecn++: good news for++, bad news for ecn over mobile," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 180–186, 2018.

[18] P. Weidenbach and J. vom Dorp, "Home router security report 2020," https://www.fkie.fraunhofer.de/content/dam/fkie/de/documents/HomeRouter/HomeRouterSecurity_2020_Bericht.pdf, 2020.

[19] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," IETF RFC 8290.

[20] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "Tack: Improving wireless transport performance by taming acknowledgments," in *Proc. ACM SIGCOMM*, 2020.

[21] lunchanddinner, "Airlink: 850 vs 500 vs 200mbps on Quest 3," https://www.reddit.com/r/OculusQuest/comments/17gwipa/airlink_850_vs_500_vs_200mbps_on_quest_3_wifi_6e/, 2024.

[22] "Speedtest Global Index – Internet Speed around the world (November 2024)," https://www.speedtest.net/global-index, 2024.

[23] V. Bajpai, S. J. Eravuchira, and J. Schönwälder, "Dissecting last-mile latency characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 5, pp. 25–34, 2017.

[24] R. Fontugne, A. Shah, and K. Cho, "Persistent last-mile congestion: not so uncommon," in *Proc. ACM IMC*, 2020.

[25] A. Tahir and R. Mittal, "Enabling users to control their internet," in *Proc. USENIX NSDI*, 2023, pp. 555–573.

[26] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma, "Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption," in *Proc. ACM SIGCOMM*, 2020.

[27] A. Bhartia, B. Chen, F. Wang, D. Pallas, R. Musaloiu-E, T. T.-T. Lai, and H. Ma, "Measurement-based, practical techniques to improve 802.11 ac performance," in *Proc. ACM IMC*, 2017.

[28] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy, "Inferring persistent interdomain congestion," in *Proc. ACM SIGCOMM*, 2018, pp. 1–15.

[29] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. USENIX NSDI*, 2015.

[30] Anurag, "What os does the router use? is it linux? - quora," https://www.quora.com/What-OS-does-the-router-use-Is-it-Linux#:~:text=Yes%20most%20of%20the%20router,the%20router%20as%20pre%2Dinstalled.

[31] "Smart Queue Management - Bufferbloat.net," https://www.bufferbloat.net/projects/cerowrt/wiki/Smart_Queue_Management/, 2017.

[32] I. Recommendations, "G.1070 : Opinion model for video-telephony applications," https://www.itu.int/rec/T-REC-G.1070, 2018.

[33] "Meeting and phone statistics – zoom help center." https://support.zoom.us/hc/en-us/articles/202920719-Meeting-and-phone-statistics, 2020.

[34] J. Zhang, H. Tong, E. Dong, X. Qian, M. Xu, X. Li, and Z. Meng, "Cold start or hot start? robust slow start in congestion control with a priori knowledge for mobile web services," in *Proc. WWW*, 2024.

[35] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *ACM Queue*, 2016.

[36] H. Meng, Y. Zhuang, Y. Noushirvani, X. Huang, and Z. Meng, "Mae: More adaptive video encoder for consistent low latency in high-quality real-time communication," in *Proc. USENIX NSDI*, 2026.

[37] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "Abc: A simple explicit congestion controller for wireless networks," in *Proc. USENIX NSDI*, 2020.

[38] "International Broadband Scorecard 2023: interactive data - Ofcom," https://www.ofcom.org.uk/phones-and-broadband/coverage-and-speeds/international-broadband-scorecard-2023-interactive-data/, 2023.

[39] Y. Xie, F. Yi, and K. Jamieson, "Pbe-cc: Congestion control via endpoint-centric, physical-layer bandwidth measurements," in *Proc. ACM SIGCOMM*, 2020.

[40] Y. Shen and Z. Meng, "Law: Towards consistent low latency in 802.11 home networks," in *Proc. USENIX NSDI*, 2026.

[41] N. Garg, "Copa congestion control for video performance," https://engineering.fb.com/2019/11/17/video-engineering/copa/, 2019.

[42] A. A. Philip, R. Athapathu, R. Ware, F. F. Mkocheko, A. Schlomer, M. Shou, Z. Meng, S. Seshan, and J. Sherry, "Prudentia: Findings of an internet fairness watchdog," in *Proc. ACM SIGCOMM*, 2024, pp. 506–520.

[43] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, 2008.

[44] A. Mishra, L. Rastogi, R. Joshi, and B. Leong, "Keeping an eye on congestion control in the wild with nebby," in *Proc. ACM SIGCOMM*, 2024, pp. 136–150.

[45] "[systemd-devel] [announce] systemd 217," https://lists.freedesktop.org/archives/systemd-devel/2014-October/024662.html#:~:text=The%20default%20sysctl.d/%20snippets%20will%20now%20set%3A, 2014.

[46] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, 1993.

[47] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, 2002.

[48] J. Renouard, "The average time spent on a website: Increase visitor engagement," website/#:~:text=The%20average%20time%20spent%20on%20a%20web%20page%20ranges%20depending,industries%2C%20is%20around%2053%20seconds., 2023.

[49] K. D. Schepper, B. Briscoe, and G. White, "Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)," RFC 9332, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9332

[50] "Gsoc2020prague - nsnam," https://www.nsnam.org/wiki/GSOC2020Prague.

[51] X. Zhang, S. Sen, D. Kurniawan, H. Gunawi, and J. Jiang, "E2e: embracing user heterogeneity to improve quality of experience on the web," in *Proc. ACM SIGCOMM*, 2019.