POSTER: User-Controllable Congestion Mitigation for Low-Latency Applications

Alexis Schlomer
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Adithya Abraham Philip
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Justine Sherry
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Zili Meng
HKUST
Hong Kong

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → **Transport protocols**; **Network measurement**;

KEYWORDS

Computer networks, Internet fairness, Internet measurement, Internet services, Real-time communications

ACM Reference Format:
Alexis Schlomer, Adithya Abraham Philip, Justine Sherry, and Zili Meng. 2024. POSTER: User-Controllable Congestion Mitigation for Low-Latency Applications. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672202.3673740

1 INTRODUCTION

Since the beginning of the decade, real-time communication (RTC) protocols, including video conferencing applications, cloud gaming, and VR/AR streaming, have rapidly expanded over the Internet. With the eventual arrival of the metaverse, RTC traffic is widely anticipated to become the predominant type of Internet traffic in the near future [4, 9].

Due to their time-sensitive nature, these real-time applications must meet strict quality of experience (QoE) standards. Users expect low latency, high frame rates, minimal freezing, and high resolution. While the average and median performance of these metrics has improved with faster network speeds, their tail distribution remains vastly insufficient for the next generation of applications [6, 7].

The causes of poor tail QoE performance are diverse, ranging from inefficiencies in the congestion feedback signaling loop [6, 9] to slow decode times at the frame reorder buffer [7]. However, this work specifically focuses on tail degradation caused by competing traffic, particularly web traffic. The multi-flow and bursty traffic patterns of web traffic are especially detrimental to real-time applications [5].

Current approaches address this problem through active queue management schemes at the router level, prioritizing latency-sensitive traffic over other types [5]. However, these solutions are challenging to deploy and require kernel support [10]. Therefore, we explore an alternative approach, examining what end-users can do independently in user-space to mitigate the effects of competing traffic.

We demonstrate that simple web browsing during a remote video conferencing call can increase the number of experienced freezes by a factor of **10x** under a moderately constrained bottleneck bandwidth setting. To alleviate this, we propose a novel multi-flow control mechanism that paces incoming HTTP traffic in user-space using the NFQueue Linux utility, preventing packet loss at the router and mitigating the detrimental effects on RTC QoE. Our approach reduces the average freeze frequency by **67%** while fully preserving the web browsing quality of experience.

2 DESIGN AND IMPLEMENTATION

To restrict the scope of our problem, we assume a single user simultaneously browsing the web and participating in an RTC meeting on two monitors connected to the same client. This setup allows us to make localized decisions more effectively, as we are the sole source of traffic and congestion at our bottleneck router. Although this scenario may not reflect all real-world conditions, it serves as groundwork for the development of more complex protocols involving multiple household members collaborating to enhance their collective QoE, which we identify as an area for future research.

Our design is based on observing that web page loads exhibit bursty traffic patterns, as the sender rapidly probes the network for bandwidth (e.g., through slow start). This leads to temporary buffer overflows at our "last-mile" bottleneck router, causing packet loss for RTC flows.

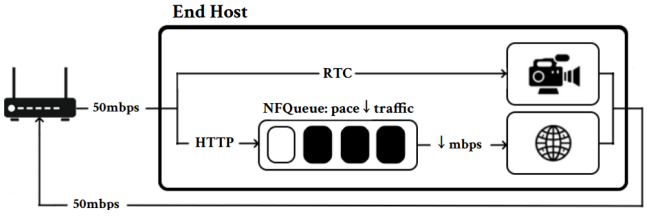

Figure 1: One user simultaneously attends an RTC meeting on one monitor and browses the web on another. We employ the NFQueue network library to pace the incoming HTTP traffic.

To address this, our core insight is to pace HTTP traffic by introducing early packet loss. This avoids buffer bloat and ensures room for RTC flows. Traditional flow-control solutions are ineffective, as

ACM SIGCOMM Posters and Demos '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0717-9/24/08.
https://doi.org/10.1145/3672202.3673740

web traffic consists of multiple distinct flows from different sources. Therefore, we implement a novel multi-flow control mechanism. This mechanism groups all incoming web traffic flows and starts dropping packets when their arrival rate exceeds a threshold.

As shown in Fig. 1, we paced ingress HTTP traffic using the NFQueue Linux network utility. We redirected all incoming TCP & QUIC traffic from ports 443 and 80 to our rate limiter, implemented in 150 lines of C++. NFQueue was chosen for its ability to run in user-space, avoiding the need for kernel changes, and offers greater extensibility and flexibility than traffic control, albeit with a slightly higher CPU load. The queue size was set to match the bandwidth-delay product (BDP) of the rate shaping, assuming a fixed RTT of 50 ms and a user-defined bandwidth. Automatically determining the optimal bandwidth limit is reserved for future work.

Finally, to test our design, we developed a custom testbed using Google Chrome for Testing [8] on two virtual monitors, controlled by Selenium [2]. We emulated our network settings with a FIFO router transmitting over a moderately constrained bottleneck bandwidth link of 50 Mbps using the traffic control [3] utility directly on the end host, with the default Linux queue size of 1000 packets.

3 EVALUATION

In this section, we first detail our experimental setup and how we achieve statistical significance before showcasing and discussing our preliminary results.

3.1 Experimental Setup

We based our analysis on Microsoft Teams as the real-time application and the Dailymail as the source of web traffic, which is a heavy website that uses multiple flows.

The key QoE metrics we focused on for RTC are average freezes per minute, specifically (1) low FPS events, defined as the frequency of FPS dropping below 10 within a one-second window, as per Zhuge [6], and (2) freezes as defined by the WebRTC standard [1], which counts deviations from a moving average of FPS over a one-second window. For web traffic, we measured page load time (PLT) as the time it takes for 95% of a page's visible region to load [8]. The webpage was loaded once every minute during the experiments.

We experimented with three scenarios: (1) the RTC service running alone, (2) alongside unpaced HTTP traffic, and (3) alongside paced HTTP traffic at different rates. To increase statistical significance, each experiment was run 30 times in a round-robin fashion, with a duration of 12 minutes per run, while excluding the first and last 30 seconds.

3.2 Preliminary Results

To highlight the detrimental impact of periodic page loads on RTC QoE, we plotted the aggregate CDFs of FPS (left) and the average number of low FPS events per minute (right) under different pacing settings in Fig. 2. In both graphs, the dashed lines represent our baseline performances.

Across all settings, the unpaced scenario performs significantly worse. Loading the Dailymail website during a Teams call increases average RTC low FPS events by over **10x** compared to the baseline. Any pacing significantly improves tail FPS, achieving up to a **67%** reduction in low FPS events from the baseline when pacing is applied from 5 Mbps to 35 Mbps. Although the effect diminishes with less aggressive pacing, it still shows a **42%** improvement over the baseline with 49 Mbps pacing on a 50 Mbps link.

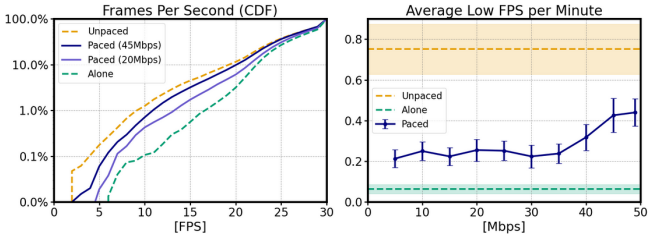

Figure 2: Aggregate CDFs and average low FPS events (i.e. CDF ≤ 10) using different pacing rates.

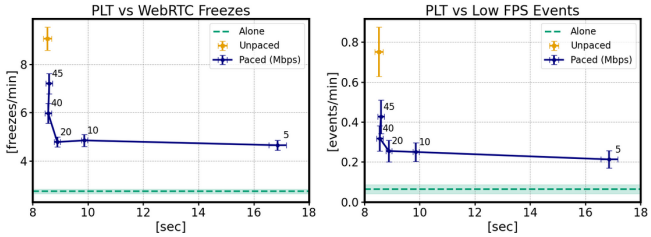

Figure 3: Trade-offs between web PLT and RTC freezes.

Next, we examine the impact of pacing HTTP ingress traffic on the page load time to identify the trade-off between QoE. In Fig. 3, we plot the achieved PLT against the number of freezes per minute as per the WebRTC standard (left) and Zhuge (right) for selected pacing scenarios. Both baselines are represented by dashed lines (RTC alone) and a special point in orange (unpaced).

To our surprise, PLT only starts to seriously deteriorate once pacing reaches 20 Mbps. Pacing above 20 Mbps remains well within the 95% CI margin of error for unpaced PLT but significantly improves RTC QoE, resulting in about a **50%** reduction in WebRTC freezes and a **67%** reduction in low FPS events at 20 Mbps.

4 CONCLUSION AND FUTURE WORK

In this work, we showed that careful pacing of incoming HTTP traffic at the end host using NFQueue can prevent packet loss of RTC flows over a moderately constrained bottleneck bandwidth FIFO router. Our approach significantly mitigates the detrimental effects on QoE for real-time applications while preserving the web browsing experience.

We establish our research agenda as follows:

(1) **Adaptive Strategies**: While our current hard-coded 20 Mbps pacing strategy suits the current setup, it won't work for all scenarios. Therefore, the end-client should periodically probe its bottleneck bandwidth to enable more adaptable strategies. Furthermore, given that transient burstiness is the main factor harming QoE of real-time flows, we should dynamically adjust the pacing and increase the bandwidth allocation for sustained competing traffic, similar to the approach used by Confucius in its per-flow bandwidth allocation scheme [5].
(2) **Increased Generalizability**: We could extend our insights to the entire household by developing a protocol that coordinates bandwidth usage to improve collective QoE and avoid conflicts at the router.

REFERENCES

[1] Identifiers for webrtc's statistics api. https://w3c.github.io/webrtc-stats/#dom-rtcinboundrtpstreamstats-freezecount.

[2] Selenium. https://www.selenium.dev/.

[3] tc(8) - linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html.

[4] 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 *Proc. USENIX NSDI*, 2018.

[5] Zili Meng, Nirav Atre, Mingwei Xu, Justine Sherry, and Maria Apostolaki. Confucius queue management: Be fair but not too fast. arXiv preprint 2310.18030, 2023.

[6] 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 *Proc. ACM SIGCOMM*, 2022.

[7] 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 *Proc. USENIX NSDI*, 2023.

[8] Adithya Abraham Philip, Rukshani Athapathu, Ranysha Ware, Fabian Francis Mkocheko, Alexis Schlomer, Mengrou Shou, Zili Meng, Srinivasan Seshan, and Justine Sherry. Prudentia: Findings of an Internet Fairness Watchdog. In *Proc. ACM SIGCOMM*, 2024.

[9] 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.

[10] Ammar Tahir and Radhika Mittal. Enabling users to control their internet. In *Proc. USENIX NSDI*, pages 555–573, 2023.