Poster: Controlling TCP Socket Queueing Delay

Hyun Hu Park
Hong Kong University of Science and Technology
Hong Kong
hhpark@connect.ust.hk

Zili Meng*
Hong Kong University of Science and Technology
Hong Kong
zilim@ust.hk

CCS Concepts

• **Networks** → **Transport protocols**; • **Software and its engineering** → *Buffering*.

Keywords

TCP socket, AQM, low-latency, real-time application

ACM Reference Format:
Hyun Hu Park and Zili Meng. 2024. Poster: Controlling TCP Socket Queueing Delay. 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/10.1145/3672202.3673745

1 Introduction

Real-time applications are tremendously emerging in recent years. From video conferencing to virtual reality and quantitative trading, real-time applications require an extremely low latency of < 100 ms or even lower [8]. Operators for decades have deployed content delivery networks, edge access nodes and numerous architectures to reduce the RTT to be as low as 10-20 ms [11]. Also, some applications (e.g., WebRTC [1]) choose to rely on UDP sockets since UDP offers more flexibility in loss recovery and congestion control.

However, compared to the TCP sockets, using UDP sockets brings considerable development and maintenance overhead to the operators. For UDP sockets, the operators need to maintain everything (e.g., congestion control, loss recovery) themselves. But when using TCP sockets, one function `send()` will handle all transport functionalities since they are all in the Linux kernel. In the meantime, when new congestion control algorithms are proposed, operators have to re-implement the algorithm again to fit the algorithm into their own frameworks. Consequently, content providers either need to maintain a separate team for the different protocol stack in WebRTC, or start to go back to TCP sockets [4, 7].

In this poster, we identify a critical factor of why TCP socket has higher latency in real-time applications – the *queueing delay at the socket send buffer*. The send buffer keeps a copy of the *in-flight* packets until they are acknowledged by the receiver, and stores data that is *pending* to send by the socket, as shown in Figure 1. However, such a design can incur a considerable latency at the send buffer. In fact, in our measurement, the queueing delay at the socket can be hundreds of milliseconds. This is because the buffer size is usually static and set by the OS (e.g., 4 MB in Linux [5]), but the amount of in-flight packets (bandwidth-delay product, BDP) can be much smaller now with the decrease of RTT. For example, when the RTT is 10 ms and throughput is 30 Mbps, the BDP is only 37.5 KB. In this case, if the send buffer is full, it takes more than 10 RTTs to drain all the data in the send buffer at the throughput of 30 Mbps. This will only become worse with the further decrease of RTT, and in these real-time applications.

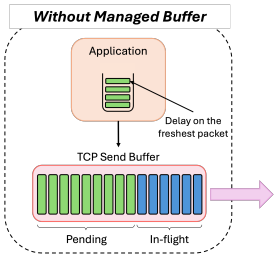

Figure 1: Current TCP buffer

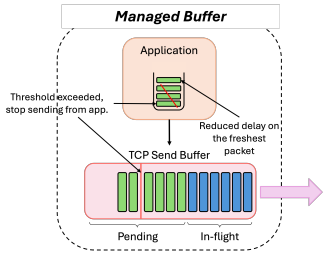

Figure 2: Buffer with queue management

The key reason is because the send buffer lacks *active management*. If the application sends too much to the socket all at same time, the buffer will be full, and the `send()` callback will either be blocked or return with the error code `EAGAIN` [2], alerting the application that the application is sending too fast. Applications such as real-time video streaming can accordingly adapt the sending rate [7]. But before the buffer is full, the application cannot easily tell how much pending data is in the buffer and how long the queueing delay will be.

Therefore, we are motivated to borrow the active queue management (AQM) in network routers [9] and enforce it to TCP socket send buffer as well. AQM is widely used in many edge routers such as WiFi routers where the queueing delay can be high[1]. Without AQM, traditional congestion control algorithms can only know the congestion when the queue on routers are full. AQM such as CoDel [9] helps to signal the congestion control algorithms to maintain a reasonable queueing delay. So our goal is to maintain the delay of the pending data in the send buffer around a target rather than setting a static buffer size.

In this poster, we propose a stragithforward method which early signals the application when the estimated queueing delay in the send buffer exceeds a threshold. As shown in Figure 1, the send buffer contains two types of packets – in-flight packets waiting to be acknowledged and pending packets waiting to be sent. The in-flight packets have already been sent out so the copy in the buffer

*Zili Meng is the corresponding author.

[1]The default AQM in Linux is fair queue CoDel [6] while more than 90% WiFi routers on the market are Linux-based [10].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACM SIGCOMM Posters and Demos '24, August 4–8, 2024, Sydney, NSW, Australia
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0717-9/24/08
https://doi.org/10.1145/3672202.3673745

does not incur any additional delay. Our goal is to minimize the amount of pending packets to control the queueing delay in the TCP socket. In this case, real-time applications could adapt their traffic rate (e.g., bitrate or framerate in video conferencing) early to enjoy a lower end-to-end latency. Also note that since we do not modify the congestion control, the throughput will still be maintained. We implement this with a preliminary prototype and show that we're possible to reduce the end-to-end latency between socket `send()` and `recv()` callbacks by up to 80% in low RTT scenarios.

2 Design

To mitigate the observed latency accumulation, we actively regulate the rate at which the application writes data to the TCP send buffer. This approach relies on monitoring the size of the occupied TCP send buffer, particularly the packets that are yet to be sent through the link. The unsent packets in the TCP send buffer, which are the *pending packets* in Figure 2, are calculated as follows.

$$\text{size of pending packets} = \text{occupied buffer size} - \text{BDP} \quad (1)$$

where BDP (bandwidth-delay product) is the size of in-flight packets in Figure 2. Since we want to minimize the queueing delay, we can estimate the queueing delay as:

$$\text{queueing delay} = \frac{\text{size of pending packets}}{\text{bandwidth}} \quad (2)$$

By dividing the equation by the bandwidth, we can derive a condition in which the potential latency caused by the unsent packets in the TCP send buffer exceeds a predetermined threshold. This condition can be expressed as:

$$\frac{\text{occupied buffer size}}{\text{bandwidth}} - \text{RTT} > \text{threshold} \quad (3)$$

In this condition, the threshold is a configurable parameter representing the maximum tolerable latency for the queueing delay at the send buffer. When the calculated value from the above condition exceeds the threshold, it indicates that the unsent packets in the buffer are likely to cause a delay beyond the acceptable limit. In this case, we take action to prevent the application from adding more data to the buffer until the queueing of the buffer is mitigated. For example, we can block the `send()` callback for sockets in the blocking mode and return `EAGAIN` error codes in the non-blocking mode. This design proactively prevents excessive latency accumulation by active monitoring of the buffer occupancy and dynamically adapts to the current network conditions such as bandwidth and RTT, allowing a responsive control of the packets sent by the application. With this design, the latency introduced to the freshest data can be flexibly controlled by adaptively adjusting the size of pending data in the send buffer by the measured bandwidth and RTT in TCP connections. This is crucial for real-time applications – video conferencing can always encode and deliver the most recent video frames, and trading applications can send the latest market index.

3 Evaluation

Setup. To quantitatively measure the implication of our design, we construct a controlled experimental environment. Using two Ubuntu Linux docker Containers (kernel version 6.6.12) interconnected via a network link. We set the RTT to 10 ms and the bandwidth to 30 Mbps using Linux tc, which are the average broadband

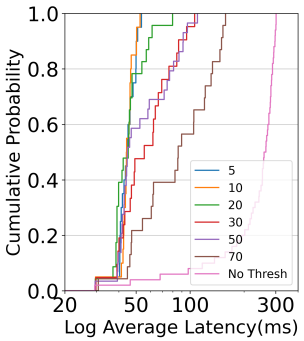

Figure 3: ECDF of log average latency

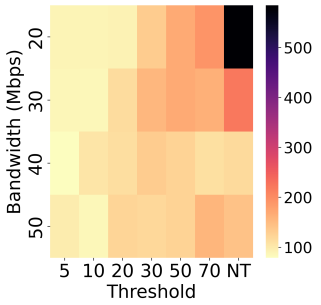

Figure 4: Average latency in various BW (in ms)

network latency by SpeedTest [3]. We later briefly sweep these parameters. The core of the experiment consists of 50 iterative data transfers of 51200-byte chunks between the server and the client with a 10 ms interval (bit rate of 41 Mbps). For each iteration, we measure the time taken to send the entire chunk and over 50 iterations of file transmission, we observe and analyse the implications of the queuing delays within the TCP send buffer.

Results. Figure 3 illustrates the distribution of average end-to-end latency for various threshold values. Without our design (`No Thresh`), the median latency can be as high as 250 ms. When we set the threshold to 5-30 ms, the median latency can be reduced to less than 50 ms, yielding a reduction of 80%. Note that even if the RTT is only 10 ms, the minimum latency is still around 30 ms, due to the transmission delay of each data chunk. Also note that, in our experiments, the throughput is always the same since the link capacity is always fully utilized.

We later sweep the bandwidth and threshold settings. We present in Figure 4 on how our queueing management strategy reacts to different bandwidths of the network. `NT (No Threshold)` exhibits the most dramatic changes in latency as bandwidth varies. The efficacy of queue management is highlighted in lower bandwidths where maximum average latency is decreased by up to 84.7% in 20 Mbps. As the bandwidth goes lower, the benefits of our algorithm is also more significant.

4 Conclusion and Future Work

This paper provides the evidence of the effectiveness of threshold-based queue management in reducing latency during TCP file transfers for application with real-time data, where data can quickly become stale. Our findings highlight how tailoring this queue management mechanism to specific network conditions, especially bandwidth, can significantly reduce the latency in the send buffer of TCP socket.

In the future, we will investigate how our algorithm performs in real-world applications and how it interacts with congestion control, with fluctuating bandwidth, and with the adaptive buffer size allocated by the Linux kernel. We will also investigate how to optimize the threshold and coordinate with the applications for the feedback signals.

References

[1] Webrtc. https://webrtc.org/.

[2] linux - write to tcp socket keeps returning eagain - stack overflow. https://stackoverflow.com/questions/36539580/write-to-tcp-socket-keeps-returning-eagain, 2016.

[3] Speedtest global index – internet speed around the world – speedtest global index. https://www.speedtest.net/global-index, 2024.

[4] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *Proc. USENIX NSDI*, 2018.

[5] Eric Dumazet. Tcp/misc works. http://vger.kernel.org/netconf2018_files/EricDumazet_netconf2018.pdf, 2016.

[6] Toke Høiland-Jørgensen, Paul McKenney, Dave Taht, Jim Gettys, and Eric Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. IETF RFC 8290.

[7] 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.

[8] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *Proc. USENIX NSDI*, 2024.

[9] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 2012.

[10] Peter Weidenbach and Johannes vom Dorp. Home router security report 2020. https://www.fkie.fraunhofer.de/content/dam/fkie/de/documents/HomeRouter/HomeRouterSecurity_2020_Bericht.pdf, 2020.

[11] Xiaokun Xu and Mark Claypool. Measurement of cloud-based game streaming system response to competing tcp cubic or tcp bbr flows. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 305–316, 2022.