# Analysis of TCP Packet Transmission Characteristics in 5G Networks

Youngho Cho, Moonbeom Kim, Sungrae Cho, and Jeongyeup Paek
Department of Computer Science & Engineering, Chung-Ang University, Seoul, Republic of Korea
{cho0h5, mbkim, srcho, jpaek}@cau.ac.kr

***Abstract*—Congestion control is an indispensable mechanism for the transmission control protocol (TCP) to provide reliable communication. Although many congestion control algorithms (CCA) have been proposed, traffic volume has increased significantly over the years due to technological advances and the growing number of radio devices, and the networks have grown larger and more complex. Inspired by this, we conduct experiments to re-investigate the performances of *CUBIC* and *BBR* in the context of recent 5G cellular networks to reveal interesting RTT fluctuations. These unexpected RTT fluctuations can negatively impact mechanisms that rely on RTT for congestion control. We analyze the causes of this phenomenon, provide insights, and briefly discuss the potential for improving CCA based on our observations.**

***Index Terms*—TCP, CUBIC, BBR, congestion control, 5G**

## I. Introduction

Transmission control protocol (TCP), which provides reliable end-to-end communication, is a widely used protocol at the transport layer for ensuring stable IP networks [1]. Particularly, the *congestion control algorithm (CCA)* of TCP is a core mechanism for maintaining network stability and optimizing performance [2]. To prevent network overload and reduce packet loss, it regulates the transmission rate by adjusting the "congestion window (cwnd)" depending on network conditions. However, in dynamically changing real networks, adapting the cwnd to an optimal value in a timely manner, depending on the network condition, is highly challenging and complex. For example, if the size of the cwnd is increased conservatively or decreased aggressively, it can result in underutilization of bandwidth, leading to low throughput and unnecessary latency. Conversely, it can exacerbate network congestion due to unnecessary retransmissions, packet loss from queue overflow, and increased delays.

Since the introduction of CCA in TCP, extensive studies have been conducted to improve performance through efficient resource utilization and enhanced congestion avoidance (as shown in Fig. 1). *Tahoe* is the first algorithm for congestion control in TCP [3]. It controls network congestion by 'slow start' and 'congestion avoidance', allowing the 'fast retransmitting' of the lost packets to improve transmission efficiency and reduce unnecessary delays. However, it has slower recovery during congestion and is vulnerable to consecutive packet losses. *Reno* has introduced the 'fast recovery' mechanism to overcome the limitation of Tahoe [4]. It can effectively respond to packet loss, enhancing network performance and stability. *CUBIC* is one of the most popular CCA and has been used as the default in Linux systems since 2006 [5], [6]. This is designed to deliver optimal performance in high-speed, high-bandwidth networks, providing fast transmission speeds and a stable connection. However, in the case of loss-based CCAs mentioned above, there is a bufferbloat issue, which can lead to high Round-Trip Time (RTT). To address these issues, Google proposed *bottleneck bandwidth and round-trip propagation time (BBR)*, a delay-based CCA [7], [8]. By continuously monitoring changes in RTT to avoid congestion, it maintains low latency while achieving stable performance.

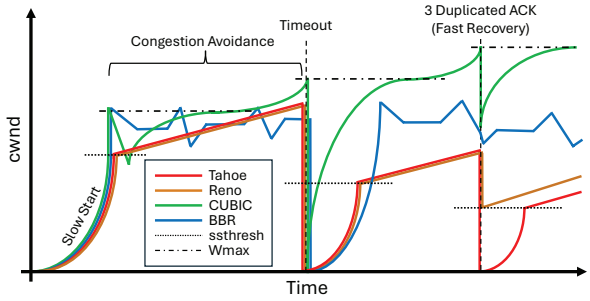
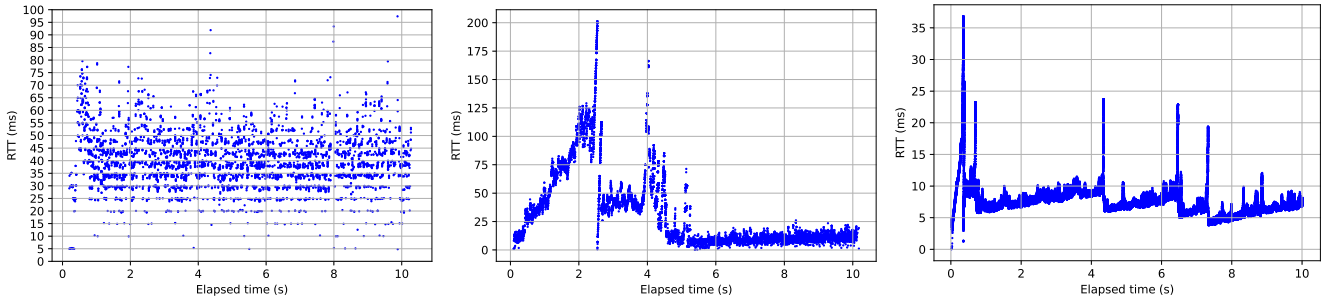



**Fig. 1:** TCP congestion control algorithms

However, with the emergence of new technologies such as 5G and the growing number of radio devices, networks are becoming larger and more complex than ever before. According to GSMA Intelligence, global mobile data traffic is expected to grow from around 100 EB per month in 2023 to 485 EB by 2030 [9]. Therefore, at this juncture, we need to re-investigate the CCA of TCP in the current network environment. In this work, we conduct experiments on the representative CCAs (*CUBIC and BBR*) in various communication environments (*5G, Wi-Fi, and Ethernet*). We observed the unexpected RTT fluctuation patterns in 5G networks, unlike in Wi-Fi and Ethernet environments. We investigate and analyze the causes of these phenomenons, and briefly discuss the potential for improving CCA to achieve higher throughput

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A4A5034130 & No. RS-2024-00359450), and also by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-RS-2022-00156353) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).




**(a)** 5G RTT **(b)** Wi-Fi RTT **(c)** Ethernet RTT

**Fig. 2:** Comparison of RTT Pattern Across 5G, WiFi, and Ethernet Connections Using CUBIC

and lower latency based on our observations at the end of the paper.

## II. Background

CCAs regulate packet transmission by determining the cwnd, which is the upper limit on the number of segments. In this section, we briefly introduce the representative congestion control mechanisms such as Tahoe, Reno, CUBIC, and BBR.

***Tahoe and Reno*** are early, simple CCAs. In Tahoe and Reno, when the cwnd is below the slow 'start threshold (ssthresh)', the cwnd doubles with each round, and this phase is called the slow start phase (*e.g.,* the first part in Fig. 1). The initial ssthresh has unlimited value, and it is set to half the cwnd whenever a congestion senses due to timeout or 3 duplicated ACK. When the cwnd exceeds the ssthresh, it enters the congestion avoidance phase. In this phase, if they successfully deliver data, they increase the cwnd by one (*e.g.,* the second part of Fig. 1). However, when the event fails, Tahoe and Reno respond differently depending on the situations. Tahoe resets the cwnd to its initial value wherever there is a timeout or 3 duplicated ACKs. Reno does the same for a timeout. However, if 3 duplicated ACKs are received, Reno reduces the cwnd by half, a process known as fast recovery (*e.g.,* the last in Fig. 1). Unfortunately, Tahoe and Reno are known to underutilize the network, especially in high bandwidth-delay produce network.

***CUBIC*** [5] increases the cwnd using a cubic function. At the time of packet loss, it saves the cwnd as 'window maximum ($W_{max}$)' and then performs a multiplicative decrease. Afterward, when increasing the cwnd, CUBIC follows the concave part of the cubic function to quickly return to $W_{max}$. Once the cwnd exceeds $W_{max}$, it follows the convex part of the cubic function, increasing the cwnd further to probe larger window sizes. CUBIC can maintain the cwnd close to $W_{max}$ for a long time, because the slope of the cubic function flattens near $W_{max}$ (dash-single dotted line in Fig. 1). This allows CUBIC to achieve high utilization on long fat networks. However, being loss-based, it tends to fill the buffers of bottleneck node, which increases queuing delay and lead to higher RTT.

***BBR*** [7], a delay-based CCA, assesses congestion through RTT rather than packet loss. BBR measures the round-trip propagation time ($RTprop$) and bottleneck bandwidth ($BtlBw$) to calculate the bandwidth-delay product (BDP = $BtlBw \times RTprop$), which is then used for congestion control. To find the maximum $BtlBw$ of the link, BBR periodically increases the sending rate. If the throughput does not increase but RTT does, BBR interprets this as a queue forming in the bottleneck buffer and subsequently reduces the sending rate to clear the queue. If this is not the case, BBR assumes that the link's condition has improved and increases the base pacing rate. In addition, every 10 seconds, BBR reduces the inflight packets for 200 ms to drain the queue and accurately measure the $RTprop$. Using the measured BDP, BBR adjusts the cwnd and paces the packets accordingly. Due to these characteristics, BBR addresses the bufferbloat issues associated with CUBIC and achieves lower RTT while maintaining similar throughput. Hence, accurate RTT measurement is crucial for BBR.

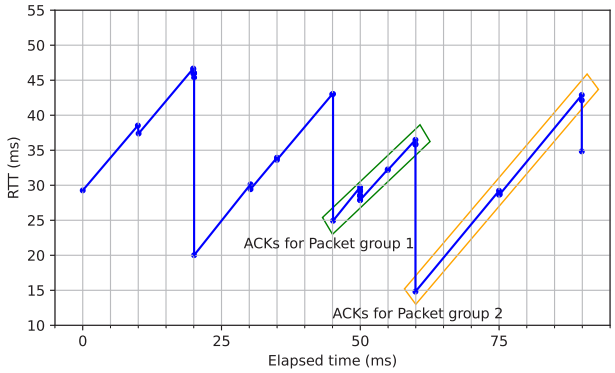
## III. Analysis

We conducted measurements of CUBIC and BBR in Ethernet, Wi-Fi, and 5G environment. We used the 5 GHz band for the Wi-Fi connection and SKT's 5G network for the experiments. We generated traffic using iPerf3. And we used an AWS EC2 m5.large instance [10] as the sender for iPerf3, which supports a maximum bandwidth of up to 10 Gbps. An iPhone was used as the receiver. And captured TCP packets using tcpdump [11] to collect data.

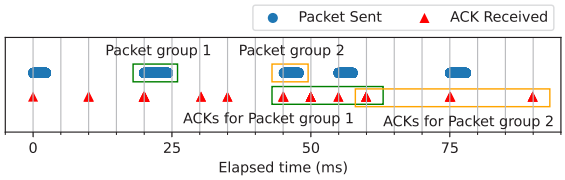
### A. TCP packet transmission characteristics with CUBIC

Fig. 2 shows the RTT characteristics measured at the server while generating traffic from the server. As shown in Fig. 2b and Fig. 2c, the RTT does not fluctuate significantly. In contrast, Fig. 2a shows that when communicating over TCP with a device using 5G, the RTT fluctuates significantly. From this figure, two key characteristics can be observed. First, the RTT tends to be close to multiples of 5 seconds. Second, this trend diminishes as the RTT increases.

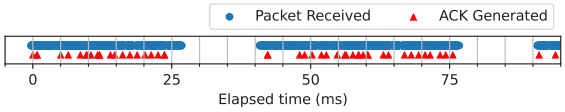
To examine this pattern more closely, we enlarged Fig. 2a and created Fig. 3a. As shown in Fig. 3a, the RTT does not simply increase randomly; rather, it follows a pattern of increasing at a consistent rate over time before suddenly dropping. This behavior is due to the scheduling characteristics of the 5G uplink and the way CUBIC transmits packets. We discovered that the base station does not allocate resources




**(a)** Close-up of 5G RTT




**(b)** Sender-Side Packet Event Timeline




**(c)** Receiver-Side Packet Event Timeline

**Fig. 3:** Close-up of RTT fluctuations (CUBIC)

whenever the device has data to send, but instead allocates them at regular intervals. To analyze why this phenomenon occurs, we plotted the times when the sender transmitted packets and when ACKs were received in Fig. 3b. In Fig. 3b, the upper-side blue dots represent the moments when the sender transmits packets, and the lower-side red triangles indicate when the server receives ACKs sent by the receiver. Additionally, both figures are overlaid with a grid at 5 ms intervals on both the x-axis and y-axis. First, by observing the red dots, we can see that ACKs arrive at the server at intervals of 5 ms, 10 ms, or 15 ms. And CUBIC tends to send packets in bursts each time on ACK arrives, causing the sending timing to synchronize with the arrival of the ACKs. Therefore, the intervals between the blue dots and the red dots also appear as multiples of 5 ms.

To analyze why packets are arriving at regular intervals, we also captured packets on the receiver-side. To capture TCP packets on the iPhone, we used rvictl [12]. By using rvictl, the iPhone's network interface can be recognized on macOS, allowing TCP packets to be captured using tcpdump in the same way as with other interfaces. The reason ACKs arrive in multiples of 5 ms is due to the scheduling characteristics of the 5G uplink not downlink. Fig. 3c shows the packets captured on the receiver-side, where the upper-side blue dots represent the moments when packets sent by the sender arrived, and the lower-side red triangles indicate when the receiver generated ACKs. Fig. 3c shows that the receiver

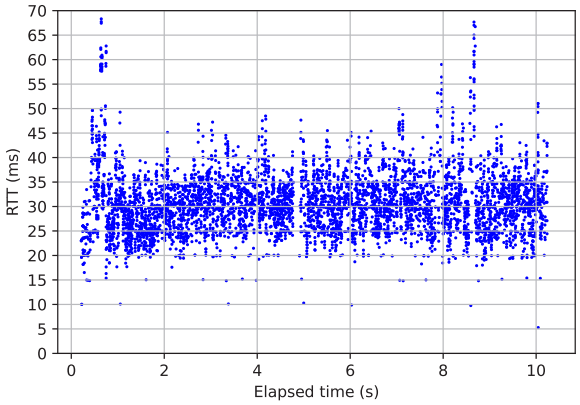


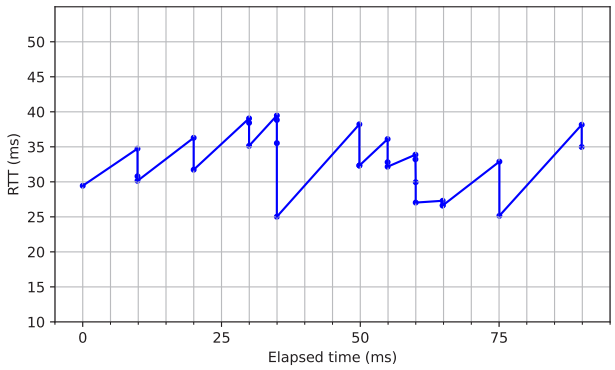

**Fig. 4:** 5G RTT with TCP BBR

receives packet independently of the 5 ms interval and also generates ACKs without regard to this timing. Therefore, it is inferred that the 5G base station allocates resources at 5 ms intervals.

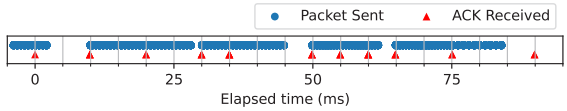
Upon closer examination, all the ACKs within the green rectangle are for the packet group 1. Second ACK in the green rectangle arrived approximately 5 ms later than first one. Because the ACK for packets that were sent almost simultaneously arrived 5 ms later, it can be observed that the RTT also increased by approximately 5 ms. Additionally, third ACK also arrived 5 ms later than second one, resulting in an approximate 5 ms increase in RTT. However, the RTT increases by slightly less than 5 ms, rather than exactly 5 ms. The reason is that although CUBIC sends packets in bursts, it cannot transmit all packets simultaneously. Therefore, the rate of RTT increase over time is slightly less than 1. The reason the ACKs that arrived simultaneously at 60 ms have significantly different RTT is because each ACK corresponds to packets sent at different times. Among the ACKs that arrived at 60 ms, the one on top is for packet group 1, while the one below is for packet group 2. For this reason, the RTT exhibits a specific pattern of increase and decrease. Although this pattern does not impact CUBIC's performance, it has been instrumental in identifying and analyzing the patterns that emerge during communication over 5G.

### *B. TCP Packet Transmission Characteristics with BBR*

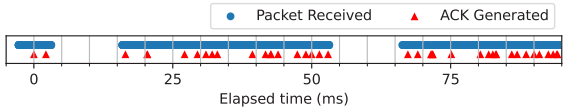
We conducted experiments to determine if this pattern also occurs with BBR, and the results are shown in Fig. 4. As seen in Fig. 4, although there are significant fluctuations in BBR as well. However, the tendency for RTT to align with multiples of 5 ms is not as apparent. This is because, as shown in Fig. 5b, BBR paces every packet. This means that, unlike CUBIC, BBR does not send packets in burst when ACKs arrive. Instead, it regulates the sending rate to transmit packets at appropriate intervals. However, despite the absence of such a pattern in Fig. 4, Fig. 5b shows that ACKs still arrive in multiples of 5 ms, due to the uplink resources being allocated at 5 ms intervals in the 5G network. And as shown in the magnified Fig. 5a, the rate of RTT increase still




**(a)** Close-up of 5G RTT




**(b)** Sender-Side Packet Event Timeline




**(c)** Receiver-Side Packet Event Timeline

**Fig. 5:** Close-up of RTT fluctuations (BBR)

remains largely unchanged. Due to BBR's pacing, the tendency for ACKs to arrive in multiples of 5 ms has diminished. However, the RTT increase rate still does not exceed 1, and the pattern of multiple ACKs arriving simultaneously at regular intervals is still observed. This increase in RTT is unrelated to the network's congestion state. By identifying this specific pattern of RTT increase and decrease, delay-based CCAs could be improved to avoid mistakenly interpreting it as network congestion.

### *C. Scheduling interval based on the transmission rate*

To verify whether the scheduling pattern of the 5G uplink is fixed or variable, we conducted additional experiment. We reversed the traffic flow, sending it from the 5G device to the EC2 instance. To measure the scheduling intervals based on the transmission rate, we developed an app using Xcode and adjusted the traffic manually, instead of using iPerf3. This app generated traffic by exponentially increasing the transmission rate from 0.0059 Mbits/sec to 109 Mbits/sec over 16 steps. We conducted the experiment by maintaining a single TCP link, sending traffic for 2 seconds in each step, followed by a 1 second pause. Fig. 6 shows the intervals between incoming packets captured at the server. This figure demonstrates that the scheduling interval is not fixed at 5 ms. Instead, as the transmission rate increases, the ACK intervals vary, ranging from 20 ms, 5 ms, and even smaller intervals.

## IV. Summary

In this paper, we observed that RTT fluctuates in a specific pattern during communication with 5G devices. This behavior

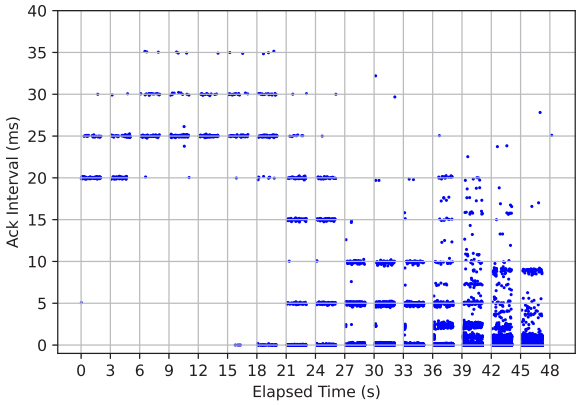



**Fig. 6:** ACK Interval According to Traffic Volume

was found to be unrelated to network congestion and was instead caused by the scheduling characteristics of the 5G uplink, which led to ACKs being transmitted in bursts every 5 ms. Additionally, we reversed the traffic flow by varying the transmission rate from the 5G device, and observed that the uplink scheduling interval is not fixed but varies depending on the transmission rate. We identified the pattern of RTT fluctuations, and we expect that this understanding will allow us to distinguish whether an increase in RTT is due to uplink scheduling characteristics or network congestion. We leave this for future work.

## References

[1] W. Eddy, "Transmission Control Protocol (TCP)," 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[2] S. Floyd, "Congestion Control Principles," 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2914

[3] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.

[4] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[5] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 64–74, jul 2008.

[6] S. Hemminger, "Make CUBIC The Default," 2006, [last accessed on August 2024]. [Online]. Available: https://github.com/torvalds/linux/commit/597811ec167fa01c926a0957a91d9e39baa30e64

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, p. 20–53, oct 2016.

[8] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *Proceedings of the Applied Networking Research Workshop (ANRW '18)*. ACM, 2018, p. 19.

[9] GSMA, "The mobile economy," 2024, [last accessed on August 2024]. [Online]. Available: https://www.gsma.com/solutions-and-impact/connectivity-for-good/mobile-economy/wp-content/uploads/2024/02/260224-The-Mobile-Economy-2024.pdf

[10] A. W. Services, "Amazon EC2 Instance types," 2024, [last accessed on August 2024]. [Online]. Available: https://aws.amazon.com/ec2/instance-types

[11] T. Group, *TCPDUMP Documentation*, 2024, [last accessed on August 2024]. [Online]. Available: https://www.tcpdump.org/index.html#documentation

[12] Apple, *Set Up iOS Packet Tracing*, 2024, [last accessed on August 2024]. [Online]. Available: https://developer.apple.com/documentation/network/recording-a-packet-trace#Set-Up-iOS-Packet-Tracing