Are Mobiles Ready for BBR?

Santiago Vargas
Stony Brook University
USA
savargas@cs.stonybrook.edu

Gautham Gunapati
Stony Brook University
USA
ggunapati@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
USA
anshul@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
USA
arunab@cs.stonybrook.edu

Abstract

BBR is a new congestion control algorithm that has seen widespread Internet adoption in recent years with an estimated 40% of Internet traffic volume as BBR traffic. While many studies examine the performance and fairness of BBR on desktops and servers, there is still a question of how BBR would behave on mobile devices. This is especially important because mobiles represent a large segment of Internet devices. In this work, we study the potential performance bottlenecks of BBR if it were to be deployed on Android devices. We compare the performance of BBR and the default congestion control algorithm Cubic for different devices and device configurations. We find that BBR performs poorly compared to Cubic, especially under low-end device configurations. Further investigation reveals that this poor performance is because of packet pacing which is enabled in BBR by default. Pacing increases the computational overhead, which can affect performance for low-end devices. To address this problem, we propose a first cut solution that modifies BBR's pacing behavior to improve performance while still retaining the benefits of packet pacing.

CCS Concepts

• **Networks** → **Transport protocols**; *Network performance analysis*; *Network measurement*.

Keywords

BBR, Mobiles, TCP Packet Pacing

ACM Reference Format:
Santiago Vargas, Gautham Gunapati, Anshul Gandhi, and Aruna Balasubramanian. 2022. Are Mobiles Ready for BBR?. In *ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3517745.3561438

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10…$15.00
https://doi.org/10.1145/3517745.3561438

1 Introduction

BBR [11] is a new congestion control algorithm that has had major adoption throughout the Internet [26]. In fact, studies showed that BBR, which stands for Bottleneck Bandwidth and RTT, was used by 40% of Internet traffic volume in 2019 [26]. Most of the research on BBR performance has focused on specific applications [23], differing network conditions [16, 21], and performance against other TCP congestion control algorithms [9, 10, 23, 35]. To the best of our knowledge, there is no published work that empirically analyzes BBR's behavior on mobile devices.

While BBR is not currently deployed on Android, this work explores the performance bottlenecks if BBR were to be rolled out to mobile devices. Given the prevalence of mobile devices and mobile traffic [4] as well as the adoption of BBR throughout the Internet, it is important to understand how BBR would behave on mobile devices.

In this paper, we provide an in-depth look at the performance of mobile devices sending traffic using BBR. We especially focus on low-end mobile device configurations because low-end mobiles are susceptible to TCP performance degradation [17], are more likely to be used in developing regions, and represent usage of the next billion first-time Internet users [7]. We study BBR, the newer BBR2 version [12–14], and Cubic (the default congestion control for Android) under different CPU settings and number of TCP connections. We specifically examine scenarios where large amounts of traffic is sent from the mobile via the uplink under a range of parallel connections. Some of these workloads may not be common to today's mobile Internet usage, but we explore them to capture potential future use-cases and to get ahead of future problems.

In these scenarios, we find that BBR and BBR2 significantly underperform Cubic when the number of parallel TCP connections increases and the device configurations move from high CPU frequencies (representing high-end phones) to low CPU frequencies (representing low-end phones). When the mobile device is run using the default CPU configuration, BBR underperforms Cubic by at least 11% in terms of goodput with as little as 1 connection. The performance difference only increases with more connections and as the CPU capacity drops. For example, under a low-end device configuration with 20 parallel connections, BBR's goodput is 55% that of Cubic.

The question then is this: *what difference between BBR and Cubic is responsible for BBR's poor performance on low-end mobiles?* While

there are several differences between BBR and Cubic, BBR's use of packet pacing is particularly distinct from Cubic which does not pace packets by default and instead sends them upon receiving acks. BBR's packet pacing helps it maintain lower RTTs and reduced loss rates since packets are less likely to be sent at a rate that network routers cannot handle.

By exploring these key differences, we uncover that TCP's packet pacing mechanism is the cause of poor BBR performance on mobiles. TCP's internal pacing sends one buffer of data at a time and sets a timer in between data buffer deliveries. However, for low-end devices, this pacing mechanism imposes substantial overhead on the TCP stack since every data-send has additional pacing overhead. In fact, enabling TCP pacing for Cubic similarly decreases Cubic's performance, thus confirming our findings.

As a first cut approach, we modify TCP's packet pacing to reduce the frequency at which packet pacing happens. As a result, TCP paces less frequently but sends more data per pacing period. Our experiments across mobile configurations demonstrate much improved BBR and BBR2 goodput, on par with Cubic's performance, while still retaining the benefits of packet pacing. Thus, since BBR requires packet pacing, modifying how TCP's pacing works can make it feasible to deploy BBR on mobile devices, even under low-end configurations and multiple connections.

2 Background on BBR

This section provides an overview of BBR and BBR2 congestion control algorithms and how they differ from Cubic.

BBR: TCP congestion control algorithms at the sender determine the number of packets that can be sent in-flight, to maximize throughput and minimize latency. Traditional, loss-based congestion controls, such as Cubic, use loss signals to gauge when the network is congested and back-off sending rates in the presence of losses. While losses can indicate that there is network congestion, a loss often occurs when router buffers are already saturated and TCP is already experiencing high delays due to congestion.

BBR is a newer congestion algorithm developed by Google in 2016 [11] that ignores losses and instead estimates bandwidth and RTT as its main signals. BBR computes the optimal network capacity [24], the bandwidth-delay project (BDP), by estimating the the network's bottleneck bandwidth and propagation delay. Then, BBR controls sending rates by setting both congestion window (cwnd) and packet pacing rates proportional to the connection BDP. The cwnd limits the maximum number of packets in-flight (unacknowledged packets) so that the network is not saturated. Packet pacing ensures that packets are not sent across the network at a rate higher than the bottleneck bandwidth to avoid filling the buffer.

BBR2: Although BBR has been largely deployed, recent studies have highlighted some of the drawbacks of BBR, including unfairness to other TCP variants, BBR's high retransmissions in shallow buffers, and BBR's poor performance when it estimates propagation delay [10, 22, 23, 33, 35]. As a result, BBR2 was introduced by Google as an attempt to address these issues [12–14]. BBR2 uses persistent losses as a signal to cut sending rates to improve fairness. Though BBR2 is actively being developed, we include BBR2 in our experiments for the sake of completeness.

Config.	Pixel 4 Freq.	Pixel 6 Freq.	Cores
Low-End	576MHz	300MHz	LITTLE
Mid-End	1.2GHz	1.2GHz	LITTLE
High-End	2.8GHz	2.8GHz	BIG
Default	Dynamic	Dynamic	Dynamic

Table 1: Different Mobile CPU configurations used on the Pixel phones to measure BBR performance.

3 Experimental Methodology

Our goal is to understand the implications of using the emerging BBR congestion control on smartphones. Past work [17] has shown that congestion control can be compute intensive for phones with low compute capacities, resulting in unexpected performance implications. To this end, our goal is to study the performance of the new BBR congestion control vs. the Android default Cubic congestion control on phones with different compute capacities. We find that the Cubic congestion control for Android is the same as the Cubic implementation in the corresponding Linux kernel.

3.1 Device Setup

We perform experiments on two Android phones, the Pixel 4 (2019 release date) and the Pixel 6 (2021 release date). The Pixel 4 and 6 run Android 11 and 12 with Linux kernel versions 4.14 and 5.10, respectively. We select these two Android phones because Android is based on Linux and more readily customizable. Additionally, the Pixel line of devices has open source Android kernels that are relatively stock, making the process of modifying the Linux kernel feasible [1].

Enabling BBR and BBR2:

While this paper explores the performance of BBR on Android, BBR is not actively deployed on Android and does not ship off-the-shelf with the phones. Since BBR is part of the Linux kernel and both Pixel devices have sufficiently new Linux kernel versions, we are able to include BBR in the devices' kernel configurations. Once BBR is enabled, we recompile the Android kernels for the mobile phones that now include BBR and flash the new kernel on the mobile device. To the best of our knowledge, we are the first work to present a performance evaluation of BBR on mobile devices.

BBR2 is slightly more challenging to deploy on the Pixel devices because it is still not part of the mainstream Linux kernel. Since the latest BBR2 version is based off of Linux kernel 5.13 and the Pixel 6's kernel is based off of version 5.10, we choose the Pixel 6 as a good candidate to run BBR2. To this end, we backport all BBR2 changes to the Pixel 6's kernel, resolve code mismatches, and recompile a Pixel 6 kernel with BBR2 enabled. We open source our port of BBR2 for the Pixel 6 kernel [5].

CPU Configurations: Our goal is to study the performance of BBR on phones under different compute configurations. To this end, we experiment with four CPU configurations as shown in Table 1 that we describe below.

We choose lower CPU frequencies on the phone to emulate low-end phones similar to past work [17]. Lower-end phones that are popular in developing regions [17] operate at clock frequencies in the range of 300-1300MHz; CPU frequencies we set for the *Low-End* configuration is in the lower end of this range. In addition, we also experiment with medium- and high-end CPU frequencies. Default refers to using the dynamic CPU scaling performed by the

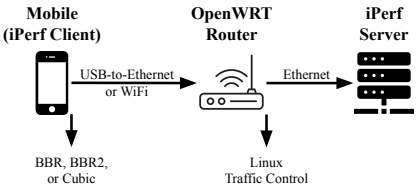


Figure 1: Illustration of our testbed topology.

phone by default. We were unable to run BBR on low-end phones because many of these lower-end devices have older kernels that are incompatible with BBR and/or do not have open source kernels that can be modified.

- **Low-End:** The userspace governor [2] is set to fix the mobile frequency at the minimum CPU frequency. We disable all BIG cores in the BIG.LITTLE CPU architecture.
- **Mid-End:** The mobile frequency is set at the median CPU frequency for the LITTLE cores. Further, we disable all BIG cores in the BIG.LITTLE CPU architecture.
- **High-End:** The mobile frequency is set at the maximum CPU frequency for the BIG cores in the BIG.LITTLE CPU architecture. Further, we disable all LITTLE cores in the BIG.LITTLE CPU architecture.
- **Default:** The default governor [2] is maintained and CPU frequency targets are not set. All BIG and LITTLE cores are enabled, and the device is left to scale frequency and decide which cores to use automatically. This device configuration represents off-the-shelf behavior that aims to balance CPU compute power and battery life.

3.2 Experiment Setup

Figure 1 depicts our overall experiment setup. In our setup, the mobile device (iPerf client) sends data via the OpenWRT router to the iPerf server.

This uplink traffic pattern (sending bulk amounts of traffic from a mobile) can be seen in large file uploads (eg. video uploads). Researchers also suggest that future Augmented Reality and Virtual Reality applications require increased uplink capacity, especially since they can be enabled using emerging technologies like 5G [18–20, 32, 34].

Since the mobile device sends data, we change the congestion control on the mobile device. Our network setup also allows network conditions to be set on the OpenWRT router using Linux traffic control (tc) [3]. By default, results are presented without any network conditions being set by tc, unless otherwise specified. Every iPerf3 result that we present is averaged over at least 10 experiment runs where iPerf3 sends data for 5 minutes.

For our network topology, we deploy two setups—an Ethernet LAN and a WiFi LAN. We use a Linksys 1900ACS router and install OpenWRT 21 on it. Further, we connect a desktop server via Ethernet to the OpenWRT router. The mobile phone is connected over Ethernet or WiFi.

Ethernet LAN: Under the Ethernet LAN setup, the mobile phone is connected to the router via Ethernet cable using a USB-to-Ethernet adapter. We verify that this setup is able to achieve close to the 1Gbps line rate (see §4). The Ethernet LAN setup represents a reliable setup where connection medium may not have a major effect on results.

WiFi LAN: Under the WiFi LAN setup, the OpenWRT router is configured as an access point. The mobile phone is the only device connected to the router via WiFi and is ~1 meter away from the router. This setup represents a more realistic network medium for average users. However, results may have increased variability due to WiFi artifacts such as interference, variable network speeds, etc.

4 Performance of BBR on Mobiles

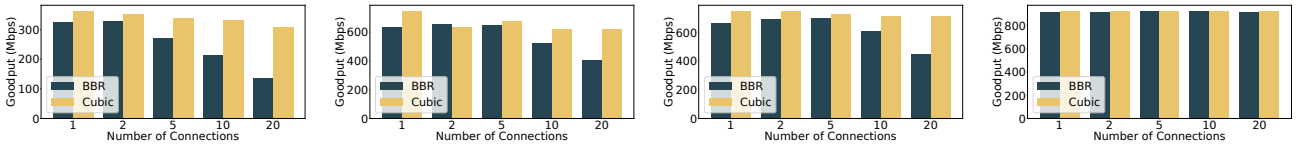
In this section, we evaluate BBR's performance on different mobile device configurations under a range of parallel TCP connections for Ethernet and WiFi. The goal is to see what BBR's performance would look like, especially for future uplink-demanding applications like AR and VR [19, 20]. The amount of uplink data being sent is uncommon in today's mobile Internet use, but our experiments are designed to capture future use-cases. For instance, 5G mmWave currently supports up to 200Mbps uplink speeds [28], and we expect that future applications will use this capacity.

Our main finding is that, under these specific conditions, BBR and BBR2 perform worse compared to Cubic especially under lower-end mobile configurations. This performance discrepancy exists across both the Pixel 4 and 6 devices. We also perform experiments on an LTE network but we do not find discrepancies between BBR and Cubic performance here. This is because the performance in LTE networks is not limited by the device configuration (i.e., the CPU capacity) but by the bandwidth (more details in Appendix A.1). While LTE networks are bandwidth limited, future 5G networks with higher bandwidths are likely to see similar BBR performance as our WiFi and Ethernet experiments.

4.1 Across Device Configurations

We begin by comparing the goodput of BBR and Cubic across different device configurations under the Ethernet LAN setup. Figure 2 shows the aggregate goodput results from running BBR and Cubic across all four configurations on the Pixel 4. We make several key observations:

- **Capable of Ideal Goodput:** Both BBR and Cubic under High-End device configurations (Figure 2d) are able to achieve at least 915Mbps goodput. Since line rate on the OpenWRT is 1Gbps, this shows that the mobiles are capable of reaching goodput near line-rate using BBR and Cubic TCP algorithms and that the network is able to support these high throughputs. However, this near line-rate throughput is only achieved when the mobiles operate at maximum frequency with BIG cores.
- **BBR Performance Drops with More Connections:** Looking at the Default, Low-End, and Mid-End device configurations in Figures 2c, 2a, and 2b, we see that BBR performance degrades significantly with more connections while Cubic goodput degrades minimally and only under the Low-End and Mid-End device configurations. For example, BBR's goodput under the Low-End configuration drops 58% from 325Mbps with 1 connection to 138Mbps under 20 connections. In comparison, Cubic's goodput between 1 and 20 connections only decreases by 15%.
- **BBR Generally Achieves Lower Goodput than Cubic:** Looking at Low-End and Default configurations, BBR achieves lower



(a) Low-End Configuration (b) Mid-End Configuration (c) Default Configuration (d) High-End Configuration

Figure 2: Average BBR and Cubic goodput for Low-End, Mid-End, Default, and High-End CPU configurations. BBR performs worse than Cubic under Low-End, Mid-End, and Default configurations with increased connections.

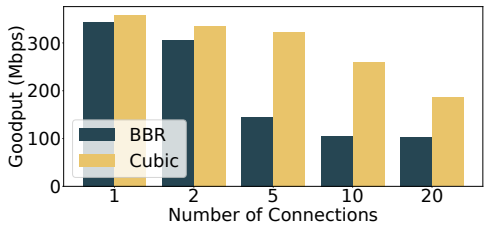


Figure 3: Average BBR and Cubic goodput for the Pixel 6 under the Low-End configuration. BBR's performance gap in comparison to Cubic increases as the number of connections increases.

goodput than Cubic irrespective of the number of connections. For Low-End mobile configuration, Cubic under 1 and 20 connection sees a 364Mbps and 310Mbps goodput whereas BBR only sees 325Mbps (11% decrease) and 138Mbps (55% decrease) for the same conditions. Similarly, the Default configuration under 1 and 20 connections shows a 14% and 37% drop, respectively, when using BBR instead of Cubic, and the Mid-End configurations shows similar drops for 10 and 20 connections.

Figure 3 shows that the BBR goodput on Pixel 6 under Low-End configuration is similar to that on Pixel 4. While Cubic's goodput degrades more than that on Pixel 4 under 20 connections, BBR's goodput is comparably 45% less than Cubic.

Main Takeaway: BBR performs worse than Cubic under Low-End, Mid-End, and Default mobile configurations, and the performance degradation is significant when there are larger numbers of parallel connections.

4.2 BBR2 Performance

Next, we perform BBR2 experiments using the WiFi LAN setup for the Pixel 6 (see §3.1) with Low-End configuration and 20 connections. We use the WiFi setup here for BBR2 since when we compile BBR2 for the Pixel 6, we find that our BBR2 kernel does not support Ethernet connections. Our BBR2 experiments (not presented here) show similar results and trends whereby Cubic still performs better than BBR2 (and BBR). From Cubic to BBR and BBR2, there is a 23% and 20% drop in goodput, respectively.

5 Dissecting BBR's Performance Gap

We now investigate why BBR and BBR2's performance is inferior compared to Cubic, especially when the number of connections increases. Both BBR versions differs from Cubic in three key aspects:

(1) **Congestion Model:** BBR recomputes a large part of its model (estimated BW and RTT) on every acknowledged packet.
(2) **Cwnd:** BBR and BBR2 set cwnd proportional to the BDP.
(3) **Packet Pacing:** BBR and BBR2 enable TCP packet pacing and set the pacing rate proportional to the bottleneck bandwidth.

Cubic, on the other hand, uses a simple AIMD logic upon every ACK'ed/lost packet and does not use packet pacing by default.

In order to facilitate our investigation, we create a master BBR kernel module that allows us to control each of these three aspects. Our module lets us disable computation performed by the BBR model, set a custom cwnd value, enable/disable packet pacing, and set specific packet pacing rates. We perform the below experiments under Low-End configuration since the performance gap is most pronounced in this setting.

5.1 Effect of BBR's Cwnd and Pacing Rates

We start by fixing cwnd and pacing rates in our module in order to check if BBR itself is artificially limiting goodput by setting small cwnd and pacing rates. We fix a cwnd value of 70 packets, similar to Cubic's average cwnd for similar iPerf experiments, and experiment with fixing pacing rates of each connection and disabling BBR's Congestion model.

5.1.1 Disabling BBR's Congestion Model: One concern is that the BBR code is more compute-intensive than Cubic's logic and may be causing excessive compute overhead, especially under poor CPU conditions. To assess this concern, we disable BBR's model update upon each acknowledged packet in *tcp_bbr.c* and repeat the fixed cwnd and pacing experiments. Consequently, BBR does not run its main code logic, and our fixed cwnd values are immediately applied every congestion loop. We find that setting Cubic-like cwnd values still results in suboptimal performance even with BBR's compute disabled.

5.1.2 Fixed Pacing Rate: We next experiment with fixing a per-connection pacing rate to get Cubic-like performance. We progressively increase the pacing rate and find that only at a pacing rate of 140Mbps per connection, the goodput of BBR reaches the goodput of Cubic (for the 20 connections case under the Low-End mobile configuration.) However, a 140Mbps pacing rate per connection is effectively unpaced and is a higher rate than the 16Mbps pacing rate that is theoretically needed per connection to achieve 315Mbps. The result is that BBR loses benefits of pacing as we show below.

5.2 Effect of Packet Pacing

5.2.1 Disabling Packet Pacing: Following from §5.1.2, we note that a high pacing rate does in fact increase BBR's performance, independent of the BBR model. We hypothesize that having an extremely high pacing rate per connection is similar to disabling packet pacing because packets are effectively bursted through the network

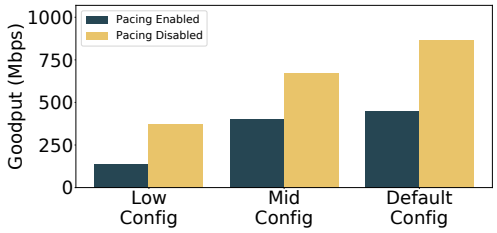


Figure 4: Effect of pacing on BBR: The goodput of BBR under Low-End configuration and 20 connections. TCP Pacing significantly decreases BBR performance.

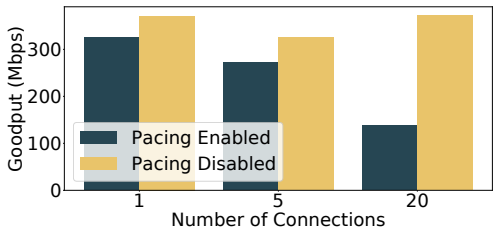


Figure 5: Effect of pacing under varying number of connections with a Low-End configuration. TCP pacing affects BBR goodput negatively under all cases, and the performance gap gets worse as the number of connection increases.

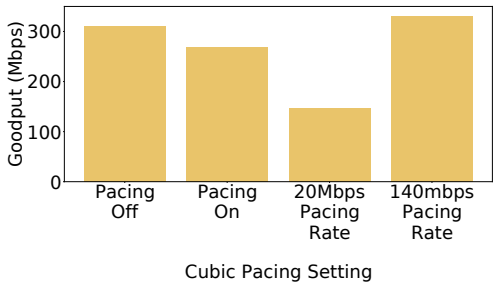


Figure 6: Effect of pacing in Cubic: Cubic goodput under Low-End configuration with no pacing (default), pacing on, and 20 and 140Mbps fixed pacing rates. TCP pacing hurts Cubic performance especially under low pacing rates.

with these excessively high pacing rates. To test our hypothesis, we disable BBR's packet pacing mechanism and allow BBR to control sending rates only through setting the cwnd.

Figure 4 shows the difference in goodput for Low-End, Mid-End, and Default configurations with and without packet pacing under 20 connections. It is evident that BBR's goodput significantly increases when packet pacing is disabled. For example, BBR's goodput under the Low-End configuration increases 2.7× when pacing is disabled. Similar trends are present in Mid-End and Default configurations, where goodput increases by 67% and 91%, respectively, when disabling packet pacing.

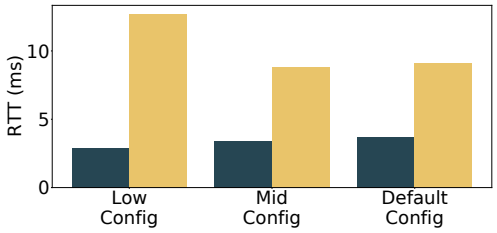


Figure 7: Benefits of pacing: The RTT of BBR under Low-End configuration and 20 connections with and without packet pacing. While pacing negatively affects goodput (Figure 4), it does help decrease RTT significantly.

Figure 5 shows that even for 1 and 5 connections, BBR's goodput increases by 14% and 19%, respectively, when pacing is disabled.

5.2.2 Is it BBR or TCP Packet Pacing?: Next, we investigate if TCP Pacing is the problem or if BBR's use of pacing is not optimal. To this end, we run experiments where we enable packet pacing in Cubic. Recall that pacing is disabled in Cubic by default. If enabled, Cubic uses TCP's internal pacing rate of $(mss * cwnd/rtt)$.

Figure 6 shows the performance of Cubic when pacing is enabled and different pacing rates are used. For these experiments, we use the Low-End configuration and 20 TCP connections. We find that when pacing is enabled, Cubic goodput also drops considerably, especially when the packet pacing rate is low. While 20Mbps should reach a maximum of 400Mbps (20Mbps × 20 connections), it only achieves 147Mbps. However, similar to BBR, when pacing is increased to 140Mbps, Cubic goodput is similar to unpaced Cubic performance. This suggests that, even for Cubic, pacing is problematic and that *TCP Pacing is not a BBR-specific problem on mobiles.*

5.2.3 Benefits of Packet Pacing: While a straw-man approach to improving BBR performance on low-end mobiles is to disable the pacing mechanism, prior work has shown that packet pacing benefits overall congestion and TCP fairness [6, 36]. For example, Figure 7 shows that RTT increases sharply for Low-End, Mid-End, and Default configurations when disabling BBR's packet pacing behavior. For all configurations, RTT more than doubles when packets are not paced, hinting at network congestion. We find that pacing is important in both the slow start and congestion avoidance phases because it reduces RTTs in both phases.

To further demonstrate this, we experiment under a 10-packet shallow buffer that is especially congestion-susceptible. While goodput increases when disabling BBR's pacing, average retransmissions increase dramatically from 37 to 13,500 packets when disabling BBR's pacing, and RTTs increase similarly to Figure 7. Since increased retransmissions and RTTs are symptoms of network congestion, BBR's pacing does in fact help combat congestion by not bursting packets.

Main Takeaway: Packet pacing under low-end mobiles and large number of connections limits BBR's performance, but pacing is necessary to prevent congestion.

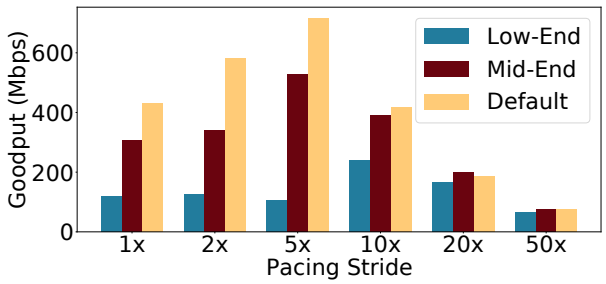


Figure 8: Goodput of Low-End, Mid-End, and Default configurations under 1×–50× pacing strides. The best pacing stride is different for the different configurations.

Pacing Stride	Skbuff Len (Kb)	Idle Time (ms)	Expected Tx (Mbps)	Actual Tx (Mbps)	RTT (ms)
1×	32.1	0.88	729	430	3.7
2×	57.7	1.54	748	580	2.2
5×	**121**	**3.22**	**751**	**717**	**1.4**
10×	120.8	5.68	426	416	1.1
20×	120.6	12.7	190	185	1.3
50×	121.4	31.1	78.1	75.6	1.4

Table 2: Socket buffer length, idle time, expected throughput, actual throughput, and RTT across different pacing strides under the Default configuration. As idle time increases, the pacing overhead decreases, resulting in improved throughput. However, as idle time increases beyond a certain point (5× in the table above), the socket buffer starts to saturate, thereby limiting throughput.

6 Improving Pacing

In this section, we present a first cut solution to improving pacing in BBR. We first go into detail on how TCP's internal pacing mechanism works. Then, we introduce the concept of a *pacing stride* that can potentially improve BBR's performance by pacing less frequently.

6.1 Background on TCP's Pacing

BBR and BBR2 by default use TCP's internal packet pacing mechanism, which in turn works by using a timer to limit the transmission of socket buffers. TCP's pacer takes as input the size of the socket buffer and the pacing rate to compute an "idle time" when packets cannot be sent:

$$idleTime = \frac{socketBufferLength}{pacingRate} \quad (1)$$

This idle sending time is implemented in the kernel as a pacing timer such that timer expiration reschedules a callback to process the socket and send the next socket buffer. Once a new data segment is sent, the timer is activated, and the connection idles from sending more data until timer expiration.

While this is an accurate pacing mechanism, setting a timer per connection for each socket buffer that is sent can cause high per-packet overheads, especially on low-end phones. The timer expiration callbacks continually reschedule connections to be processed.

6.2 Changing Pacing Stride

To regulate the pacing overhead while still sending data at the same average pacing rate, we experiment with changing the *pacing stride*. We change pacing stride by scaling *idleTime* so that the CPU paces more packets but less frequently. We include pacing stride as a scaling variable by redefining *idleTime* as follows:

$$idleTime = idleTime \times pacingStride \quad (2)$$

Since the goal is to reduce CPU pacing overhead, we experiment with strides greater than 1, specifically $pacingStride = \{1, 2, 5, 10, 20, 50\}$.

Figure 8 shows the effect of different pacing strides on Low-End, Mid-End, and Default configurations. Increasing the pacing stride significantly improves performance of BBR across all configurations compared to default BBR (i.e., $pacingStride = 1$). For example, the goodput under Default configuration increases from around 400Mbps to over 700Mbps, and that under Low-End increases from below 140Mbps to 240Mbps. We find that a pacing stride of 5× provides the best goodput for Mid-End and Default configurations and 10× provides the best goodput for the Low-End configuration. Further, Table 2 shows that keeping a pacing stride also maintains a low RTT, unlike completely disabling pacing.

To further analyze these improvements, Table 2 shows the effect of pacing strides on average socket buffer length and average idle time. Here we perform one iPerf3 run for each pacing stride and sample idle time and socket buffer length for each individual TCP pacing period (e.g., one socket buffer is sent per TCP pacing period).

We see that increasing the pacing stride also increases the average socket buffer length and average idle time. While idle time naturally increases from Eq. (2), socket buffer length increases since more packets accumulate per pacing period. However, after a certain stride length (e.g., 20× for Low-End and 5× for Default), the buffer length plateaus and is limited by the socket buffer capacity. This in turn limits the goodput as stride length is further increased.

This can be seen by comparing the expected and actual (as reported by iPerf3) throughput values in Table 2. Expected throughput is modeled as a purely pacing-limited scenario:

$$expectedTx = \frac{socketBufferLength \times 20\ connections}{idleTime}$$

For example, under the Default configuration, the actual throughput never reaches the expected value for 1× and 2× pacing stride, because of pacing overheads. For 5× onwards, the pacing is infrequent enough to mitigate CPU overheads, resulting in actual throughput more closely matching the expected. A similar trend is seen for Low-End configuration.

Therefore, there is an optimal stride length for which pacing happens infrequently enough to mitigate the CPU overhead of pacing while ensuring that the stride is not too large to risk socket buffer saturation. This optimal length will depend on at least the network conditions and the mobile device configuration; we plan to investigate the optimal stride length in detail as part of future work.

Main Takeaway: Pacing at bigger strides with more data allows BBR to achieve significantly higher throughput under CPU-constrained configurations while maintaining low RTTs and avoiding network congestion.

7 Discussion

7.1 Pacing Strides Limitations and Future Work

The pacing stride solution we outline in this paper is a first cut approach to solving the pacing problem in BBR. Below, we discuss the implications of the solution in terms of memory usage, optimality, TCP fairness, and hardware pacing alternatives.

7.1.1 Memory Usage: The pacing strides approach may increase memory usage as packets have to wait longer before they are sent. To explore this we run experiments with the Low-End configuration and 20 connections and measure RAM usage on the mobile. We find that memory is unaffected when using pacing strides.

7.1.2 Optimality: In this work, we explore 6 discrete pacing strides of 1×, 2×, 5×, 10×, 20×, 50×. However, choosing an optimal pacing stride in terms of bandwidth will depend on the mobile configuration, number of connections, network workload, and system load. We leave further exploration of the optimal pacing stride to future work.

7.1.3 TCP Fairness and Congestion: Pacing strides change the behavior of TCP pacing by including more data during a data-send but doing so less frequently. Since previous studies have show that packet pacing improves fairness [6, 36], pacing strides may increase the unfairness of BBR. Also, pacing strides may potentially cause transient congestion, delay, and loss. We need further studies to explore both fairness and congestion when using pacing strides.

7.1.4 Hardware Pacing: BBR's authors have suggested that BBR may benefit from fine-grained hardware pacing in the future [15]. This could be a viable alternative to the pacing strides approach we outline in this paper.

7.2 Device Capabilities

One main question is whether low-end phone capabilities will improve significantly by the time BBR rolls out in Android. To explore, we enumerate the device capabilities of phones at the $60 price point (similar price to previous work on low-end phones [17]) found on Flipkart, a popular e-commerce site in India. We find that the phones in this price range have on an average 4 cores, 1.31Ghz max CPU frequency and run Android version 8.

These device configurations are similar to the phones in the $60 price range found in India four years ago [17]. While the core count and max CPU frequency are similar, the Android version has increased from Android 6 to Android 8. This suggests that while newer low-end phones still have the same hardware specifications as older phones, they do indeed run newer versions of Android. This trend was also reported in previous work which finds that while phone OSes continue to improve, compute capacity lags behind [30].

8 Related Work

In this section, we discuss work related to low-end mobile performance and the BBR congestion control.

Low-End Mobile Performance: There have been a number of studies recently that examine quality-of-experience (QoE) of Internet applications on low-end smartphones. Low-end mobiles are more prevalent especially in developing countries and there is a need for device-specific changes to overcome the bottlenecks [7]. Recent studies have shown a poor web browsing performance due to the memory bottlenecks in low-end mobiles [29, 31]. Other studies have measured QoE of Internet applications on low-end mobiles and identified CPU as a main cause of QoE degradation [17]. Following this line of work, we identify the bottleneck in BBR performance on low-end mobile configurations and discuss how to alleviate this performance bottleneck.

BBR Performance: There have also been a number of studies on BBR performance, but not on mobile devices. For example, a line of work looks at BBR on WiFi and cellular networks [8, 21]. BBR performance and fairness have been studied under various non-mobile settings [9, 10, 35] with some works predicting that BBR and BBR2 will become more prevalent in the Internet [27]. Other works have compared BBR and Cubic on satellite networks and non-traditional settings [16, 25]. A more recent work looks at BBR performance in WLAN settings and exposes pacing controls to user-space to increase the pacing rate [21].

9 Conclusion

This work presents the first measurement-driven performance evaluation of BBR and BBR2 on different mobile device configurations. We find that BBR and BBR2 underperform relative to Cubic under many CPU configurations and a large number of connections. Our investigation reveals that it is the TCP pacing mechanism used by BBR that is primarily responsible for the poor goodput. While removing pacing significantly improves BBR's performance on mobile devices, the lack of packet pacing causes high RTTs and retransmissions. Instead, we suggest changes to TCP's packet pacing whereby data-sends include more data but occur less frequently, thus mitigating the overhead of frequent pacing. Our TCP pacing changes significantly improve BBR performance on mobile devices while retaining the benefits of packet pacing.

10 Acknowledgements

We thank the reviewers and our shepherd, Ethan Katz-Bassett, for their wonderful feedback. This work was supported in part by NSF grant CNS-1909356.

References

[1] [n.d.]. Building Kernels: Android Open Source Project. https://source.android.com/setup/build/building-kernels
[2] [n.d.]. CPU Governors. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
[3] [n.d.]. TC - Linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html
[4] [n.d.]. What percentage of internet traffic is mobile? [Mar '22 UPD]. https://www.oberlo.com/statistics/mobile-internet-traffic
[5] 2022. BBR2 Port for Pixel 6 Android kernel. https://github.com/SBUNetSys/BBR2-on-Android.
[6] Amit Aggarwal, Stefan Savage, and Thomas Anderson. 2000. Understanding the performance of TCP pacing. In *Proceedings of IEEE INFOCOM (Infocom '00, Vol. 3)*. IEEE, 1157–1165.
[7] Sohaib Ahmad, Abdul Lateef Haamid, Zafar Ayyub Qazi, Zhenyu Zhou, Theophilus Benson, and Ihsan Ayyub Qazi. 2016. A View from the Other Side: Understanding Mobile Phone Characteristics in the Developing World. In *Proceedings of the Internet Measurement Conference* (Santa Monica, California, USA)

(IMC '16). Association for Computing Machinery, New York, NY, USA, 319–325. https://doi.org/10.1145/2987443.2987470

[8] Eneko Atxutegi, Fidel Liberal, Habtegebreil Kassaye Haile, Karl-Johan Grinnemo, Anna Brunstrom, and Ake Arvidsson. 2018. On the use of TCP BBR in cellular networks. *IEEE Communications Magazine* 56, 3 (2018), 172–179.

[9] Ayush, Jingzhi, Melodies, Sean, Raj, and Ben. 2021. Conjecture: Existence of Nash Equilibria in Modern Internet Congestion Control. In *5th Asia-Pacific Workshop on Networking* (Shenzhen, China, China) *(APNet '21)*. Association for Computing Machinery, New York, NY, USA, 37–42. https://doi.org/10.1145/3469393.3469397

[10] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 130–136. https://doi.org/10.1145/3355369.3355579

[11] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring Bottleneck Bandwidth and Round-Trip Propagation Time. *Queue* 14, 5 (oct 2016), 20–53. https://doi.org/10.1145/3012426.3022184

[12] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Ian Swett, Victor Vasiliev, Bin Wu, and Matt Mathis Van Jacobson. IETF-105 : iccrg, Jul 2019. *BBR v2: A Model-based Congestion Control*. https://www.ietf.org/proceedings/105/slides/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00

[13] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, and Matt Mathis Van Jacobson. IETF-106 : iccrg, Nov 2019. *BBR v2: A Model-based Congestion Control*. https://www.ietf.org/proceedings/106/slides/slides-106-iccrg-update-on-bbrv2-00

[14] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, and Matt Mathis Van Jacobson. IETF-104 : iccrg, Mar 2019. *BBR v2: A Model-based Congestion Control*. https://www.ietf.org/proceedings/104/slides/slides-104-iccrg-an-update-on-bbr-00

[15] Yuchung Cheng and Neal Cardwell. 2016. Making linux TCP fast. In *Netdev conference*.

[16] Saahil Claypool, Jae Chung, and Mark Claypool. 2021. Measurements Comparing TCP Cubic and TCP BBR over a Satellite Network. In *18th Annual Consumer Communications & Networking Conference (CCNC '21)*. IEEE, 1–4. https://doi.org/10.1109/CCNC49032.2021.9369602

[17] Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian, Samir R. Das, and Michael Ferdman. 2018. Impact of Device Performance on Mobile Internet QoE. In *Proceedings of the Internet Measurement Conference* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3278532.3278533

[18] Tony Driscoll, Suzanne Farhoud, Sean Nowling, et al. 2017. Enabling mobile augmented and virtual reality with 5G networks. *AT&T: Dallas, TX, USA* (2017).

[19] Moinak Ghoshal, Imran Khan, Qiang Xu, Z. Jonny Kong, Y. Charlie Hu, and Dimitrios Koutsonikolas. 2022. NextG-up: A Tool for Measuring Uplink Performance of 5G Networks. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services* (Portland, Oregon) *(MobiSys '22)*. Association for Computing Machinery, New York, NY, USA, 638–639. https://doi.org/10.1145/3498361.3539694

[20] Moinak Ghoshal, Z. Jonny Kong, Qiang Xu, Zixiao Lu, Shivang Aggarwal, Imran Khan, Yuanjie Li, Y. Charlie Hu, and Dimitrios Koutsonikolas. 2022. An In-Depth Study of Uplink Performance of 5G MmWave Networks. In *Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases* (Amsterdam, Netherlands) *(5G-MeMU '22)*. Association for Computing Machinery, New York, NY, USA, 29–35. https://doi.org/10.1145/3538394.3546042

[21] Carlo Augusto Grazia, Natale Patriciello, Martin Klapez, and Maurizio Casoni. 2020. BBR+: improving TCP BBR Performance over WLAN. In *International Conference on Communications (ICC '20)*. IEEE, 1–6. https://doi.org/10.1109/ICC40277.2020.9149220

[22] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *25th International Conference on Network Protocols (ICNP '17)*. IEEE, 1–10. https://doi.org/10.1109/ICNP.2017.8117540

[23] Per Hurtig, Habtegebreil Haile, Karl-Johan Grinnemo, Anna Brunstrom, Eneko Atxutegi, Fidel Liberal, and Åke Arvidsson. 2018. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. In *30th International Teletraffic Congress (ITC '18, Vol. 01)*. IEEE, 218–226. https://doi.org/10.1109/ITC30.2018.00040

[24] L. Kleinrock. 1979. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *International Conference on Communications (ICC '79, Vol. 3)*. IEEE, 43.1.1–43.1.10.

[25] Feng Li, Jae Won Chung, Xiaoxiao Jiang, and Mark Claypool. 2018. TCP CUBIC versus BBR on the Highway. In *Passive and Active Measurement (PAM '18)*. Springer International Publishing, 269–280.

[26] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The Great Internet TCP Congestion Control Census. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 45, 24 pages. https://doi.org/10.1145/3366693

[27] Ayush Mishra, Wee Han Tiu, and Ben Leong. 2022. Are we heading towards a BBR-dominant Internet?. In *Proceedings of the Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 13 pages. https://doi.org/10.1145/3517745.3561429

[28] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 610–625. https://doi.org/10.1145/3452296.3472923

[29] Usama Naseer, Theophilus A. Benson, and Ravi Netravali. 2021. WebMedic: Disentangling the Memory-Functionality Tension for the Next Billion Mobile Web Users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (Virtual, United Kingdom) *(HotMobile '21)*. Association for Computing Machinery, New York, NY, USA, 71–77. https://doi.org/10.1145/3446382.3448652

[30] Javad Nejati, Meng Luo, Nick Nikiforakis, and Aruna Balasubramanian. 2020. Need for Mobile Speed: A Historical Study of Mobile Web Performance. In *4th Network Traffic Measurement and Analysis Conference* (Berlin, Germany) *(TMA '20)*. IFIP. http://dl.ifip.org/db/conf/tma/tma2020/tma2020-camera-paper37.pdf

[31] Ihsan Ayyub Qazi, Zafar Ayyub Qazi, Theophilus A. Benson, Ghulam Murtaza, Ehsan Latif, Abdul Manan, and Abrar Tariq. 2020. Mobile Web Browsing under Memory Pressure. *SIGCOMM Comput. Commun. Rev.* 50, 4 (oct 2020), 35–48. https://doi.org/10.1145/3431832.3431837

[32] Tech Qualcomm. 2018. VR And AR Pushing Connectivity Limits.

[33] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a Deeper Understanding of TCP BBR Congestion Control. In *IFIP Networking Conference and Workshops (IFIP '18)*. IFIP, 1–9. https://doi.org/10.23919/IFIPNetworking.2018.8696830

[34] Stefano Sorrentino. [n.d.]. 5G connected cars changing automotive experiences - ericsson. https://www.ericsson.com/en/blog/2021/10/powering-connected-cars-with-5g

[35] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/3355369.3355604

[36] David Wei, Pei Cao, and Steven Low. 2006. TCP pacing revisited. In *Proceedings of IEEE INFOCOM (Infocom '06, Vol. 2)*. IEEE, 3.

A Appendix

A.1 Cellular Experiments

In addition to experiments on Ethernet and WiFi, we also conduct a small set of experiments on cellular networks. Our cellular setup is similar to Figure 1 except the phone connects to the Internet via T-Mobile's LTE network in order to reach our iPerf server. We purchased a prepaid unlimited plan from T-Mobile to perform these experiments on Pixel 6.

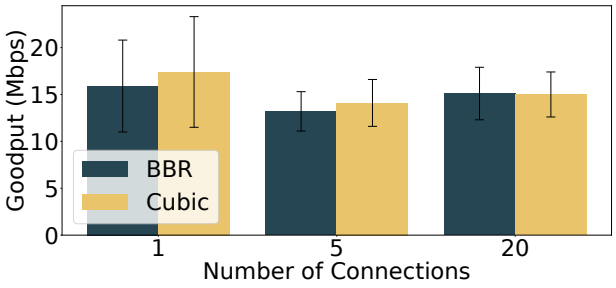


Figure 9: The goodput of BBR under Low-End configuration over cellular. BBR and Cubic performance is similar under all settings (unlike results over WiFi and Ethernet where Cubic outperformed BBR.) We find that this is because in cellular networks, the setting is bandwidth-limited and not CPU capacity-limited.

Figure 9 shows the cellular results. There is no significant difference in performance between BBR and Cubic in this setting. This is because the cellular uplink experiments are bandwidth-limited (less than 20Mbps of goodput) and do not reach sufficient levels to hit a pacing bottleneck on the mobile devices. However, recent work on mmWave 5G suggests that cellular uplinks can reach up to 200Mbps [28] which will provide sufficient network capacity. In this case, the capacity limitation and the pacing problems will become significant, similar to the WiFi and Ethernet case.

A.2 Ethics

This work does not raise any ethical issues.