# Towards a Deeper Understanding of TCP BBR Congestion Control

Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle
Chair of Network Architectures and Services, Technical University of Munich
{scholzd|jaeger|schwaigh|raumer|fgeyer|carle}@net.in.tum.de

***Abstract*—In 2016, Google published the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm. Unlike established loss- or delay-based algorithms like CUBIC or Vegas, BBR claims to operate without creating packet loss or filling buffers. Because of these prospects and promising initial performance results, BBR has gained wide-spread attention. As such it has been subject to behavior and performance analysis, which confirmed the results, but also revealed critical flaws.**

**Because BBR is still work in progress, measurement results have limited validity for the future. In this paper we present our publicly available framework for reproducible TCP measurements based on network emulation. In a case study, we analyze the TCP BBR algorithm, reproduce and confirm weaknesses of the current BBR implementation, and provide further insights. We also contribute an analysis of BBR's inter-flow synchronization behavior, showing that it reaches fairness equilibrium for long lived flows.**

***Index Terms*—TCP, Congestion Control, BBR, Reproducible Measurements**

## I. Introduction

TCP BBR is a congestion-based congestion control algorithm developed by Google and published in late 2016 [1]. In contrast to traditional algorithms like CUBIC [2] that rely on loss as indicator for congestion, BBR periodically estimates the available bandwidth and minimal round-trip time (RTT). In theory, it can operate at Kleinrock's optimal operating point [3] of maximum delivery rate with minimal congestion. This prevents the creation of queues, keeping the delay minimal.

Service providers can deploy BBR rapidly on the sender side, as there is no need for client support or intermediate network devices [1]. Google already deployed BBR in its own production platforms like the B4 wide-area network and YouTube to develop and evaluate BBR [1] and provided quick integration of BBR with the Linux kernel (available since version 4.9). This spiked huge interest about benefits, drawbacks and interaction of BBR with alternatives like CUBIC. The research community has started to formalize and analyze the behavior of BBR in more detail. While the initial results published by Google have been reproducible, demonstrating that BBR significantly improved the bandwidth and median RTT in their use cases, weaknesses like RTT or inter-protocol unfairness have been discovered since (e.g. [4, 5, 6]). As a consequence, BBR is actively improved [5]. Proposed changes usually aim to mitigate specific issues, however they need to be carefully studied for unintended side effects.

ISBN 978-3-903176-08-9 ©2018 IFIP

We introduce a framework for automated and reproducible measurements of TCP congestion avoidance algorithms. It produces repeatable experiments and is available as open source at [7]. The use of emulation using Mininet allows the framework to be independent of hardware, enabling other research groups to easily adapt it to run their own measurements or replicate ours.

We demonstrate the capabilities of our framework by inspecting and analyzing the behavior of BBR in different scenarios. While the throughput used for our measurements is orders of magnitude lower compared to testbeds utilizing hardware, we verify the applicability of our results by reproducing measurements of related work. Beyond reproduction, we deepen the analysis of BBR regarding inter-flow unfairness and inter-protocol fairness when competing with TCP CUBIC flows. Lastly, we use measurements to analyze the inter-flow synchronization behavior of BBR flows.

This paper is structured as follows: Section II presents background to TCP congestion control. In Section III, we describe our framework for reproducible TCP measurements. We performed various case studies with the analysis of BBR. The results are used to validate our framework by reproducing and extending measurements from related work in Section IV. Our BBR inter-flow synchronization analysis is discussed in Section V. Related work is presented in Section VI before we conclude with Section VII.

## II. TCP Congestion Control

Congestion control is required to achieve high network utilization for multiple flows, claiming a fair share, while preventing overloading the network with more packets than can be handled. Buffers are added to counteract packet drops caused by short lived traffic peaks, increasing network utilization. When buffers remain non-empty ("static buffers"), they add delay to every packet passing through the buffer, coined *bufferbloat*. Static buffers originate mainly from two factors, as shown by Gettys and Nichols [8]: poor queue management and failure of TCP congestion control. Algorithms like TCP NewReno [9] or TCP CUBIC [2] use packet loss as indication of congestion. However loss only occurs when the buffers are close to full at the bottleneck (depending on the queue management used). The congestion is only detected when the bottleneck is already overloaded, leading to large delays hurting interactive applications.

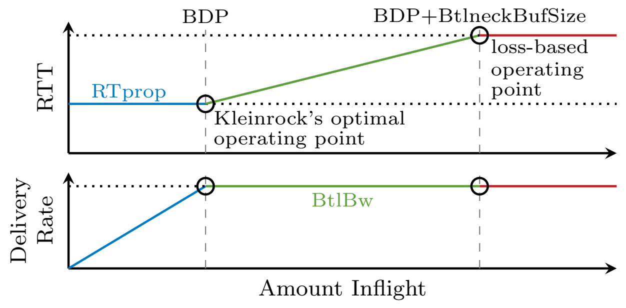


Figure 1: Effect of increasing inflight data on the RTT and delivery rate. Based on [1].

Various TCP congestion control algorithms were developed to improve on loss-based congestion control. Examples include TCP Vegas [10], adapting delay as indicator, or TIMELY [11] based on precise RTT measurements. However, these are suppressed when competing with loss-based algorithms. Hock et al. present TCP LoLa [12], primarily focusing on low latency. Hybrid algorithms using both loss and delay as congestion indication were proposed such as TCP Compound [13]. Alizadeh et al. proposed Data Center TCP (DCTCP) [14], which requires support for Explicit Congestion Notification (ECN) in network switches.

### A. TCP Optimal Operation Point

Any network throughput is limited by the segment with the lowest available bandwidth on the path. It is called bottleneck, as it limits the total throughput of the connection. Thus for modeling congestion control, a complex network path can be modeled by a single link. The delay of that link is set to the sum of all propagation delays in each direction and the bandwidth is set to the bottleneck's (BtlBw). This preserves the round trip propagation delay (RTprop). The bandwidth-delay product (BDP) as $\text{BtlBw} \cdot \text{RTprop}$ describes the amount of data that can be inflight (non-acknowledged) to fully utilize the network path and is coined Kleinrock's optimal point of operation [3].

Figure 1 visualizes the effects of an increase in inflight data on the connection's bandwidth and RTT. If less data than the BDP is inflight, there is no congestion and the RTT equals RTprop (application bound). The delivery rate corresponds directly to the sending rate, but hits the maximum when the inflight data reaches the BDP at Kleinrock's point. Increasing the inflight further causes packets to arrive faster at the bottleneck than they can be forwarded. This fills a queue, causing added delay which increases linearly with the amount inflight (recognized by delay-based algorithms). The queue is full when the amount inflight hits $\text{BDP} + \text{BtlneckBufSize}$. After this point, the bottleneck buffer starts to discard packets (recognized by loss-based algorithms), capping the RTT. This shows that both delay and loss-based algorithms operate beyond Kleinrock's optimal operating point.

### B. Bottleneck Bandwidth and Round-trip Propagation Time

The following describes basics of BBR that are important for our evaluation. Our deliberations are based on the version presented by Cardwell et al. [1] and we refer to their work for a detailed description of the congestion control algorithm or [4] for a formal analysis.

*1) Overview:* The main objective of BBR is to ensure that the bottleneck remains saturated but not congested, resulting in maximum throughput with minimal delay. Therefore, BBR estimates bandwidth as maximum observed delivery rate BtlBw and propagation delay RTprop as minimum observed RTT over certain intervals. Both values cannot be measured simultaneously, as probing for more bandwidth increases the delay through the creation of a queue at the bottleneck and vice-versa. Consequently, they are measured separately.

To control the amount of data sent, BBR uses *pacing gain*. This parameter, most of the time set to one, is multiplied with BtlBw to represent the actual sending rate.

*2) Phases:* The BBR algorithm has four different phases [15]: Startup, Drain, Probe Bandwidth, and Probe RTT.

The first phase adapts the exponential **Startup** behavior from CUBIC by doubling the sending rate with each round-trip. Once the measured bandwidth does not increase further, BBR assumes to have reached the bottleneck bandwidth. Since this observation is delayed by one RTT, a queue was already created at the bottleneck. BBR tries to **Drain** it by temporarily reducing the pacing gain. Afterwards, BBR enters the **Probe Bandwidth** phase in which it probes for more available bandwidth. This is performed in eight cycles, each lasting RTprop: First, pacing gain is set to 1.25, probing for more bandwidth, followed by 0.75 to drain created queues. For the remaining six cycles BBR sets the pacing gain to 1. BBR continuously samples the bandwidth and uses the maximum as BtlBw estimator, whereby values are valid for the timespan of ten RTprop. After not measuring a new RTprop value for ten seconds, BBR stops probing for bandwidth and enters the **Probe RTT** phase. During this phase the bandwidth is reduced to four packets to drain any possible queue and get a real estimation of the RTT. This phase is kept for 200 ms plus one RTT. If a new minimum value is measured, RTprop is updated and valid for ten seconds.

## III. TCP Measurement Framework

The development of our framework followed four requirements. **Flexibility** of the framework should allow to analyze aspects of TCP congestion control, focusing on but not limited to BBR. The **Portability** of our framework shall not be restricted to a specific hardware setup. **Reproducibility** of results obtained via the framework must be ensured. Given a configuration of an experiment, the experiment itself shall be repeatable. All important configuration parameters and the results should be gathered to allow replicability and reproducibility by others. The complete measurement process shall be simplified through **Automation**. Via configuration files and experiment description, including post processing of data and

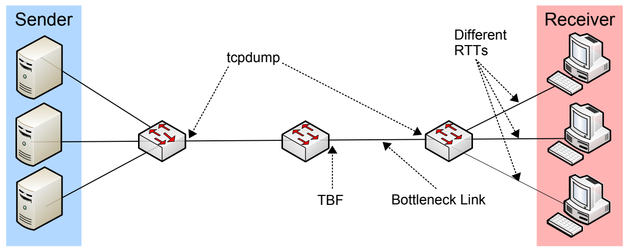


Figure 2: Mininet setup with sending and receiving hosts and bottleneck link.

generation of plots, the experiment should be executed without further user interaction.

### A. Emulation Environment

Our framework uses emulation based on Linux network namespaces with Mininet. Linux network namespaces provide lightweight network emulation, including processes, to run hundreds of nodes on a single PC [16]. A drawback is that the whole system is limited by the hardware resources of a single computer. Thus we use low bandwidths of 10 Mbit/s for the different links in the studied topology. By showing in Section IV that our measurements yield similar results as related work performing measurements beyond 10 Gbit/s, we argue that the difference in throughput does not affect the validity of the results.

### B. Setup

**Topology:** As a TCP connection can be reduced to the bottleneck link (cf. Section II-A), our setup uses a dumbbell topology depicted in Figure 2. For each TCP flow a new host-pair, sender and receiver, is added for simplified collection of per-flow data. Both sides are connected via three switches. The middle switch acts as the bottleneck by performing traffic policing on its interface. The two additional switches allow capturing the traffic before and after the policing. Traffic from the receivers to the senders is not subject to rate limiting since we only send data from the senders and the returning acknowledgment stream does not exceed the bottleneck bandwidth, assuming symmetric bottleneck bandwidth.

**Delay Emulation:** We use NetEm to add flow specific delay at the links between the switch and the respective receivers to allow configurable RTTs. This approach introduces problems for higher data rates like 10 Gbit/s where side effects (e.g. jitter) occur [4], but works well for the data rates we use.

**Rate Limit & Buffer Size:** We use Linux's Token-Bucket Filter (TBF) for rate limiting and setting the buffer size. TBFs also allow a configurable amount of tokens to accumulate when they are not needed and the configured rate can be exceeded until they are spent. We set this *token bucket size* to only hold a single packet, because exceeding the bottleneck bandwidth even for a short time interferes with BBRs ability to estimate the bottleneck bandwidth correctly [1].

### C. Workflow

Each experiment is controlled using a configuration file describing the flows. For each flow, the desired TCP congestion control algorithm, start time in relation to previous flow, RTT, and runtime have to be specified. The runtime of an experiment consists of a negligible period to set up Mininet, as well as the actual experiment defined by the length of the running flows. The framework automatically extracts data and computes the implemented metrics.

### D. Metric Collection

For each TCP flow we gather the sending rate, throughput, current RTT, and the internal BBR values. We also sample the buffer backlog of the TBF every 40 ms. As a result of one experiment, a report containing 14 graphs visualizing the metrics over time is automatically generated. A sample experiment report, including its configuration file, can be found with our source code publication [7].

We capture the headers up to the TCP layer of all packets before and after the bottleneck using `tcpdump`. The raw data is processed to generate the metrics listed below. Existing tools like Wireshark (including the command line tool `tshark`) and `tcptrace` did not meet all our requirements for flexibility. Instead we wrote our own analysis program in Python.

**Sending Rate & Throughput:** We compute the per flow and total aggregated sending rate as the average bit-rate based on the IP packet size in 200 ms intervals, using the capture before the bottleneck. The throughput is computed equal to the sending rate, but is based on the capture after the bottleneck to observe the effect of the traffic policing.

**Fairness:** We follow the recommendation of RFC 5166 [17] and use Jain's Index [18] as fairness coefficient based on the sending rate to indicate how fair the bandwidth is shared between all flows. For $n$ flows, each of them allocating $x_i \geq 0$ of a resource,

$$\mathcal{F} = 1/n \cdot [\Sigma_{i=1}^{n} x_i]^2 / \Sigma_{i=1}^{n} x_i^2$$

is 1 if all flows receive the same bandwidth and $1/n$ if one flow uses the entire bandwidth while the other flows receive nothing. The index allows quantifying the fairness in different network setups independent of the number of flows or the bottleneck bandwidth. Graphs displaying the fairness index in the remaining part of this paper are restricted to the interval $[1/n, 1]$ unless mentioned otherwise.

**Round-trip Time:** RTT values are aggregated in intervals of 200 ms and averaged to provide better stability. Samples of retransmitted packets are ignored.

**Retransmissions:** We count retransmissions of TCP segments in the packet capture before the bottleneck. We use these as an indicator for packet loss in our evaluation.

**Inflight Data:** Refers to the number of bytes sent but not yet acknowledged. We obtain this value by computing the difference of the maximum observed sequence and acknowledgment numbers in the capture before the bottleneck. This metric is only useful when there are no retransmissions.

**BBR Internal Values:** BBR keeps track of the estimated bottleneck bandwidth and RTT as well as the pacing and window gain factors. We extract these values every 20 ms using the `ss` tool from the `iproute2` tools collection.

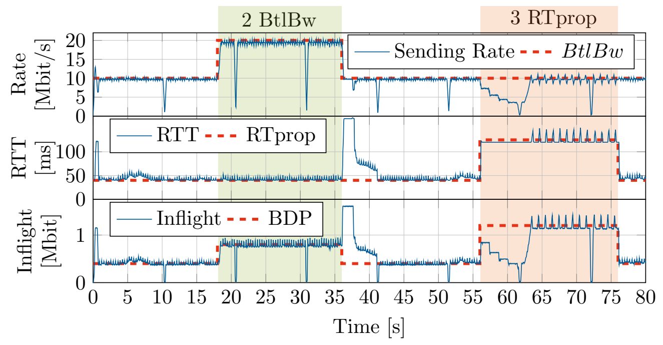


Figure 3: Single BBR flow (40 ms, 10 Mbit/s bottleneck) under changing network conditions. Values sampled every 40 ms.

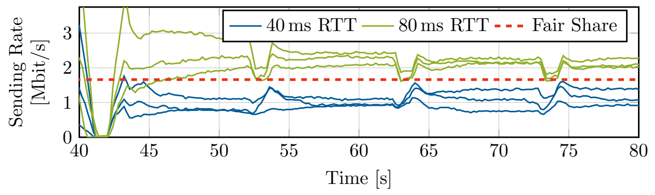


Figure 4: RTT unfairness for multiple flows with two groups of RTTs

### E. Limitations

Due to resource restriction on a single host emulated network, we are limited to bandwidths in the 10 Mbit/s range. However, our methodology provides sufficient accuracy compared to measurements utilizing real hardware. This is because both approaches use the same network stack, i.e., the same implementation of the BBR algorithm.

The CPU and RAM capacities of the test system limit the number of emulated hosts and therefore flows. We encountered problems when spawning more than 30 hosts simultaneously using a host equipped with an Intel Core i5-2520M CPU @ 2.50 GHz and 8 GB RAM.

## IV. Reproduction & Extension of Related Work

We validate the accuracy of our framework by using it to reproduce the results of related work that were based on measurements with hardware devices. The results show that the behavior of TCP BBR at bandwidths in the Mbit/s range is comparable to the behavior at higher ranges of Gbit/s. In the following, we present our reproduced results with a mention of the respective related work.

We focus on the results of two research groups. Cardwell et al., the original authors of BBR, have described their current research efforts towards BBR 2.0 [1, 5]. Goals are reduced loss rate in shallow buffers, reduced queuing delay and improved fairness among others. Hock et al. evaluated BBR in an experimental setup with 10 Gbit/s links and software-based switches [4]. They reproduced intended behavior of BBR with single flows, but also showed cases with multiple flows where BBR causes large buffer utilization.

For all following figures the raw data, post-processed data and source code to generate the figures can be found with our source code publication [7]. Unless representing a single flow, measurements were repeated five times and standard deviations are shown where applicable.

### A. Single Flow

Figure 3 shows how a single BBR flow reacts to changes of the bottleneck bandwidth in a network. Thereby, the first 55 seconds are our reproduction of [1, Fig. 3]. For equal network conditions, no significant differences are visible. The sending rate, measured RTT and inflight data closely follow the doubling in BtlBw. After the bandwidth reduction, a queue is generated, as indicated by the increased RTT estimation, and drained in the following five seconds.

Instead of an additional bandwidth reduction, we tripled RTprop at the 56 s mark. The results are surprising at first. Similar to a decrease in BtlBw, BBR cannot adapt to an increase in RTprop immediately, since the minimum filter retains an old, lower value for another 10 s. When RTprop grows, the acknowledgments for the packets take longer to arrive, which increases the inflight data until the congestion window is reached. To adapt, BBR limits its sending rate, resulting in lower samples for BtlBw. As soon as the BtlBw estimate expires, the congestion window is reduced according to the new, lower BDP. This happens repeatedly until the old minimum value for RTprop is invalidated (at approx. 62 s). Now, BBR learns about the new value and increases the sending rate again to match BtlBw with exponential growth.

While this behavior is not ideal and can cause problems, the repercussions are not severe for two reasons. First, even though the sending rate drops, the inflight data does not decrease compared to before the RTT increase. Second, it is unlikely that such a drastic change in RTT happens in the Internet in the first place.

The RTT reduction at 76 s is adapted instantly because of the RTT minimum filter.

Figure 3 also validates that our framework can sample events detailed enough ($\triangle t = 40\,\mathrm{ms}$), as both Probe Bandwidth (small spikes) and Probe RTT phases (large spikes every 10 s) are displayed accurately. However, in general we use $\triangle t = 200\,\mathrm{ms}$ for less overhead.

### B. RTT Unfairness

The RTT unfairness of BBR is visualized in [6, Fig. 1]. Two flows share a bottleneck of 100 Mbit/s, one flow having a larger RTT than the other (10 ms and 50 ms). The flow with shorter RTT starts three seconds before the other. We set the bandwidth to 10 Mbit/s and adapted all other parameters. Our reproduced results (not shown) only differ slightly: The larger flow receives about 10 % less of the bandwidth.

As shown in Figure 4, the behavior can also be observed when increasing the number of flows. Flows with equal RTT converge to a fair share within their group, however, groups with higher RTT claim a bigger share overall.

### C. Bottleneck Overestimation for Multiple Flows

BBR overestimates the bottleneck when competing with other flows, operating at the inflight data cap [4]. The analysis

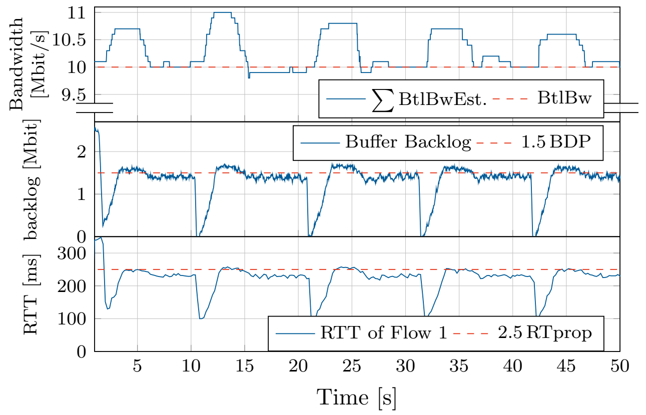


Figure 5: BDP overestimation for five flows with a 100 ms RTprop and 10 Mbit/s bottleneck (5 BDP buffer)

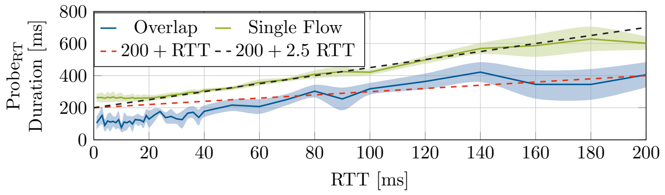


Figure 6: Overlapping Probe RTT phase duration of 5 flows

of Hock et al. predicts 2 BDP $\leq \sum_i \text{inflight}_i < 2.5$ BDP. Our experiments using a large enough buffer size of 5 BDP reproduce the results of this formal analysis as shown in Figure 5. For five simultaneously started BBR flows, the sum of the BBR estimations of BtlBw exceeds the real BtlBw after each Probe RTT phase, increasing the estimation towards the inflight cap. The backlog of the bottleneck buffer is kept at 1.5 BDP resulting in a total of 2.5 BDP.

*1) Insufficient Draining of Queues During Probe RTT:* To measure the correct RTprop value, all flows need to simultaneously drain the queue in Probe RTT. Figure 6, displaying the duration of the Probe RTT phase for five flows and the overlap thereof, shows that this is not the case. For RTTs below 40 ms the overlap is only half of the duration of the Probe RTT phase (200 ms + RTT). This is because all flows enter Probe RTT at slightly different times even though the flows are synchronized. As a consequence, the queue is not drained enough and BBR overestimates the bottleneck.

For high RTTs the overlap exceeds the theoretic maximum of 200 ms + RTT. Indeed, the duration of the Probe RTT phase for each individual flow equals 200 ms + 2.5 RTT. This is because when the previous RTprop value expires, triggering the Probe RTT phase, BBR chooses the newest measured RTT as RTprop [15]. As this value, however, is based on a measurement outside of the Probe RTT phase, it is influenced by the 2.5 BDP overestimation. As a consequence, the Probe RTT phase is longer, reducing the performance of BBR.

*2) Retransmissions for Shallow Buffers:* BBR is susceptible to shallow buffers as it overestimates the bottleneck, not recognizing that the network is strongly congested, since packet loss is not interpreted as congestion. Cardwell et al. have shown that BBR's goodput will suffer if the buffer cannot hold the additional 1.5 BDP [5].

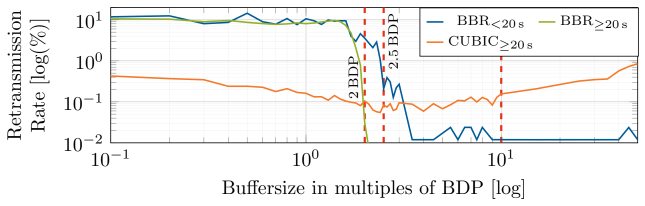


Figure 7: Retransmissions per second for 5 simultaneously started flows with different bottleneck buffer sizes

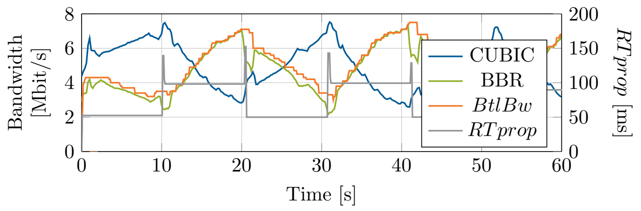


Figure 8: Competing BBR and CUBIC flow

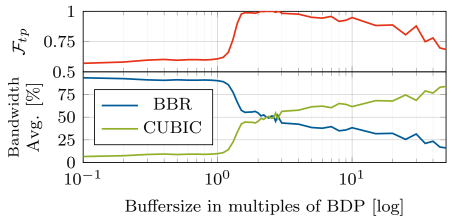
We reproduced this effect by analyzing the relation between bottleneck buffer size and caused retransmissions for both BBR and CUBIC (cf. Figure 7). Five TCP flows are started simultaneous and share a 10 Mbit/s, 50 ms bottleneck (BDP = 500 kbit). We compute the retransmission rate for different buffer sizes at the bottleneck for BBR and CUBIC individually. BBR$_{<20\,s}$ shows the first 20 s of the test including only startup and synchronization phases. BBR$_{\geq 20\,s}$ and CUBIC$_{\geq 20\,s}$ represent steady-state operation after 20 s.

For shallow buffers up to 2 BDP retransmission for BBR exceeds the amount for CUBIC by a factor of 10. This is a consequence of the buffer overestimation, in contrast to CUBIC's adaption of the congestion window for loss events. Between 2 BDP and 2.5 BDP loss only occurs during the startup and synchronization phases (before 20 s) for BBR. This is because of the initial aggressive bandwidth claiming. For even larger buffers BBR is not susceptible to loss.

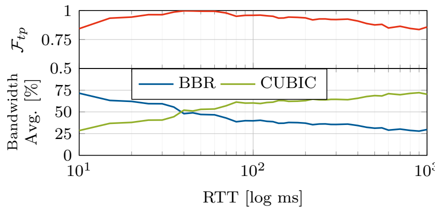
CUBIC, as loss-based algorithm, produces loss with all buffer sizes during congestion avoidance phase. However, for small buffer sizes it is a factor of 10 below BBR. Only when exceeding 10 BDP = 5 Mbit a rise in retransmissions is visible for CUBIC. This is because of taildrop, increasing the repercussions of a single loss event. However, buffers with this large capacity are not realistic in the Internet [8] and therefore only pose a theoretic problem.

## *D. Inter-protocol Behavior With CUBIC*

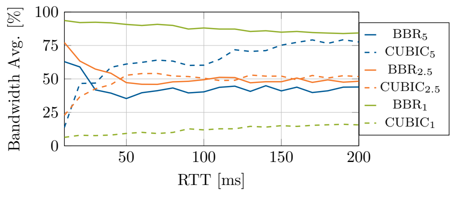
In the best case, a competing BBR and CUBIC flow reach an oscillating steady-state [5]. This is caused by the RTprop estimation of BBR as shown in Figure 8. CUBIC's aggressive probing for bandwidth causes the queues to fill up, resulting in BBR to measure a higher delay, increasing its BDP. In turn, this causes packet loss, resulting in reduced data inflight for CUBIC. Once the queue is drained, CUBIC starts to probe again, while BBR measures the correct RTprop value. This oscillation results in $\mathcal{F}$ being constantly low, however, both



(a) Increasing buffer with 50 ms RTT



(b) Increasing RTT with 2.3 BDP buffer



(c) One flow 50 ms RTT for 1, 2.5 and 5 BDP buffer

Figure 9: One CUBIC vs. one BBR flow for changing network conditions

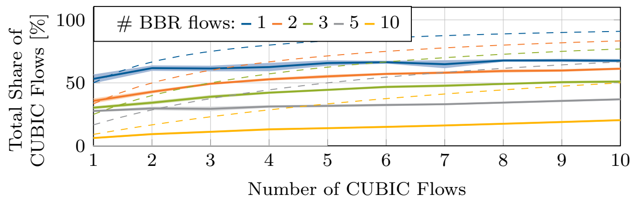


Figure 10: Bandwidth share of different number of CUBIC and BBR flows competing. Dashed lines show fair share.

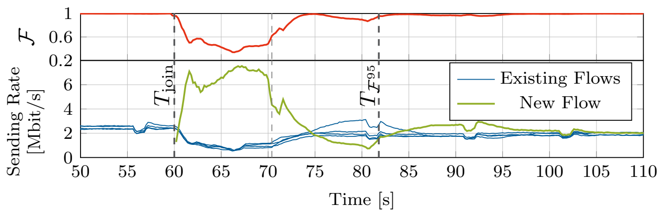


Figure 11: BBR inter-flow synchronization behavior

flows reach an equal average throughput. For the following analysis related to the inter-protocol behavior we use $\mathcal{F}_{tp}$ as fairness index based on the average throughput.

The size of the bottleneck buffer is crucial for the fairness between competing BBR and CUBIC flows [1, 4]. Figure 9a shows our reproduction of this result, displaying the bandwidth share and fairness for one BBR and one CUBIC flow for different bottleneck buffer sizes. Up to 1.5 BDP buffer size, BBR causes constant packet loss as explained in the previous section. CUBIC interprets this as congestion signal and reduces its sending rate. Up to 3 BDP both flows reach a fair share, while for further increasing buffer sizes CUBIC steadily claims more. The reason is that CUBIC fills up the ever growing buffers. For BBR this results in ever growing Probe RTT phases, i.e., reduced sending rate. The length of and the gap between Probe Bandwidth phases increases too, reducing BBR's ability to adapt. However, these buffer sizes pose only a theoretical problem (cf. Section IV-C2).

While showing the same overall behavior, RTT changes have a smaller influence on the fairness if applied to both flows as shown in Figure 9b. For all tested RTTs the fairness remained above 80 %. However, when fixating one flow at 50 ms RTT and varying the RTT of the other flow, unfairness emerges (Figure 9c). For small RTTs or shallow buffers BBR suppresses CUBIC for the already discussed reasons. In the other cases, the bandwidth share remains independent of the RTT. Only when having large buffers, CUBIC gains increasing shares with increasing RTT. Our conclusion is that the fairness between CUBIC and BBR largely depends on the botteneck buffer size, while the RTT only has a small impact.

Lastly, we evaluate how the number of flows competing with each other influences the throughput share per congestion avoidance algorithm. Figure 10 shows that CUBIC is suppressed independent of the number of flows in a scenario with 50 ms RTT and 2.5 BDP bottleneck buffer. A single BBR flow claims more bandwidth than its fair share already when competing against two CUBIC flows. In fact, independent of the number of BBR and CUBIC flows, BBR flows are always able to claim at least 35 % of the total bandwidth.

## V. Inter-flow Synchronization

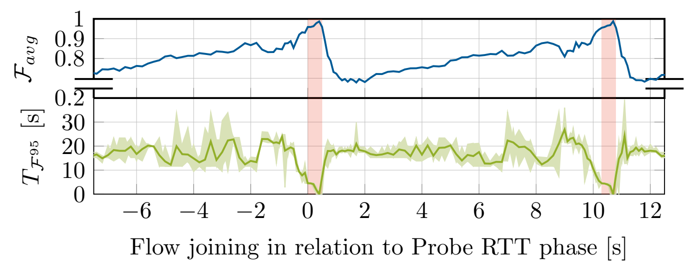
Different BBR flows synchronize themselves to avoid faulty estimations, e.g., when one flow probes for bandwidth causing a queue to form at the bottleneck, while another probes for RTT. In contrast to loss-based algorithms, this does not correlate with congestion, as the flows are impervious to loss.

### A. *Theory & Questions*

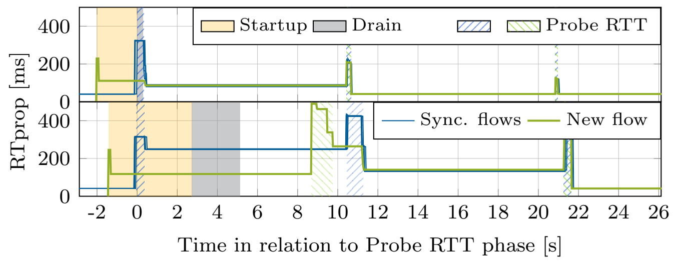
Cardwell et al. demonstrate in [1, Fig. 6] how different BBR flows synchronize whenever a large flow enters the Probe RTT phase. We visualize the process in Figure 11 with one new flow joining four already synchronized flows. The new flow immediately overestimates the bottleneck link and claims a too large share of the bandwidth. 10 s later it enters Probe RTT. The flow with bigger share drains a large portion of packets from the queue, which results in all other flows measuring a better RTprop estimate. Consequently, the flows are synchronized as the RTprop samples of all flows expire at the same time, causing them to enter Probe RTT together at the 81 s mark. Considering the fairness, it takes approximately 35 s after the new flow joined until equilibrium is reached.

To maximize performance, BBR should only spend 2% of time in Probe RTT [1, 15]. Therefore, new flows have trouble to measure the correct RTprop as active flows likely probe for more bandwidth and create queues. It causes the new flow to overestimate the BDP, inducing queuing delay or packet loss.

This raises two questions regarding the synchronization behavior of BBR flows: Is there an optimal and worst moment regarding the time until equilibrium is reached for a single flow to join a bottleneck containing already synchronized BBR flows? And secondly we want to determine if constantly



(a) Join during different times of the Probe RTT cycle. Red area marks Probe RTT phases.



(b) Correlation between Startup/Drain and Probe RTT for joining 2 s and 1.7 s before next Probe RTT phase

Figure 12: Single BBR flow joining synchronized BBR flows

adding new flows can result in extended or accumulated unfairness.

## B. Synchronization Metrics

To quantify the impact of a new flow joining we use two metrics based on Jain's fairness index $\mathcal{F}$. For better comparison we define $T_{\text{join}}$ as the point in time when the flow of interest, i.e. the last flow, has joined the network (cf. Figure 11). As first metric, we define $T_{\mathcal{F}^{95}}$ as the point after $T_{\text{join}}$ for which $\mathcal{F}$ remains stable above 0.95, i.e. no longer than 2 s below this threshold. Second, we compute the average fairness $\mathcal{F}_{\text{avg}}$ in the interval $[T_{\text{join}}, T_{\text{join}} + 30\,\text{s}]$.

In the following we analyze the behavior of flows with equal RTTs. We assume that all effects described in the following will scale similarly as described in Section IV-B with RTT unfairness between flows.

## C. Single Flow Synchronization Behavior

To analyze the basic synchronization behavior, we use the scenario of one new BBR flow joining a network with four other BBR flows already synchronized and converged to a fair share. Figure 12a shows our experimental evaluation when joining a new flow in relation to the Probe RTT phase of the synchronized flows.

As expected, a periodic behavior is revealed, with the best case for a new flow to join being during the Probe RTT phase. It synchronizes immediately as the queues are drained and the new flow can measure the optimal RTT, leading to low $T_{\mathcal{F}^{95}}$ and high $\mathcal{F}_{\text{avg}}$. The worst case is if the flow joins directly after the other flows left the Probe RTT phase. At this point, the queue is building again as the flows keep 2 BDP inflight, resulting in the new flow severely overestimating the BDP. It remains in this state until the old flows enter Probe RTT again (up to 10 s later), draining the queue and synchronizing with the new flow. This behavior of aggressively taking bandwidth from existing flows can be harmful when many short living BBR flows join, leading to starvation of long-living flows.

In general, it lasts 20 s until $T_{\mathcal{F}^{95}}$ is reached, but the later the new flow joins during the cycle, the higher varies $T_{\mathcal{F}^{95}}$ (10 to 30 s). The local optimum when joining 2 s before the Probe RTT phase with $T_{\mathcal{F}^{95}} = 10\,\text{s}$ is because the existing flows enter the Probe RTT phase while the new flow drains after the Startup as shown in Figure 12b. Consequently, all flows drain the queue and measure a new optimal RTprop, synchronizing immediately, yet overestimating the bottleneck because the queue created during Startup is not entirely drained yet. In contrast, the worse case directly afterwards (1.7 s before next Probe RTT) with $T_{\mathcal{F}^{95}} = 22\,\text{s}$ is caused by the existing flows entering Probe RTT, draining the queue, while the new flow is in Startup. This causes the new flow to drastically overestimate the bottleneck until leaving Startup, suppressing other flows.

Considering the prevalence of short-lived flows in the Internet [19, 2], this high $T_{\mathcal{F}^{95}}$ value poses a significant disadvantage of TCP BBR. Initially, flows during this time suppress other flows through unfair bandwidth claims, which is only solved when reaching a fair share.

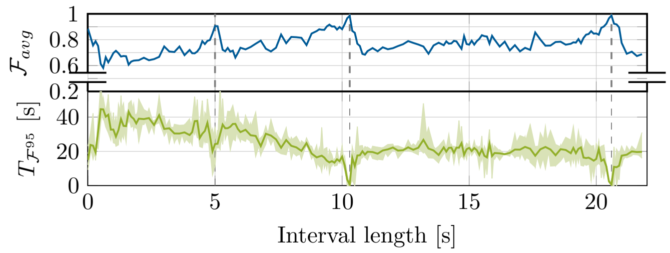
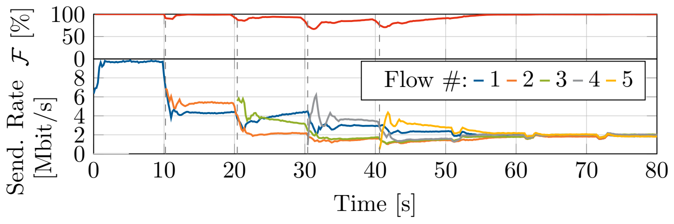


Figure 13: Different join intervals for subsequent flows

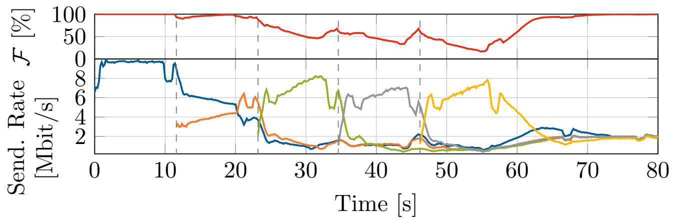
## D. Accumulating Effects

To evaluate if negative effects of multiple flows joining can accumulate, i.e. whether the duration of unfairness can be prolonged, we change the scenario to have a new flow join every $x$ seconds up to a total of five BBR flows (cf. Figure 13).

Optima are visible for intervals matching the duration of the Probe RTT phase of the already active flows at approximately 10 s and 20 s. When all flows join at the same time, they all measure a good RTprop value within the first few packets, synchronizing them immediately. For intervals smaller than 10 s accumulating effects are visible as new flows rapidly join, not allowing the fairness to stabilize. As for a single flow, $T_{\mathcal{F}^{95}}$ and $\mathcal{F}_{\text{avg}}$ improve with increasing interval. For flows joining every 5 s an additional local optimum is visible as every second flow joins during the Probe RTT phase of the other flows. For intervals larger than one Probe RTT cycle (after flows leave Probe RTT, approximately 10.5 s), $T_{\mathcal{F}^{95}}$ and $\mathcal{F}_{avg}$ show the behavior for a single flow joining. This is because all prior



(a) 10.1 s join interval (during Probe RTT)



(b) 10.5 s join interval (immediate after Probe RTT)

Figure 14: Identified best/worst case join intervals

flows have already synchronized, resulting in them already converging towards an equilibrium before the next flow joins.

Analyzing the effect of a new flow joining on individual existing flows, e.g. the longest running flow, is difficult for the lack of a good metric. We therefore select the best and worst case join intervals displayed in Figure 14 for a visual analysis. As the minimum value of $\mathcal{F}$ depends on the number of flows ($1/n$), it is normalized using percentages.

Figures 14a and 14b show the effects of subsequent flows joining during (best case) or immediately after (worst case) the Probe RTT phase. Similar to the effects on the last flow joining, existing flows are only influenced by the timing of the next flow joining. Within the group of synchronized flows, they converge to their fair share. The synchronization itself depends on the timing and happens at most after 10 s. The resulting unfairness is only caused by the new flow. The overall time until bandwidth equilibrium is approximately 55 s and 70 s, respectively. We attribute the 15 s difference to the longer synchronization phase in the latter case (10 s) and bigger unfairness thereof.

Throughout all our tests we encountered rare cases where $T_{\mathcal{F}^{95}}$ extended for up to 50 s, which are not reproducible. We attribute this instability to the sensibility of the join timing.

Summarizing, the fair sharing of bandwidth is intertwined with the timing of new flows joining the network. Except during the brief Probe RTT phase, equilibrium is only reached after 20 s and can extend up to 30 s. However, there are no effects accumulating beyond the interval of one Probe RTT phase. The timing only has a short term effect on the amplitude of unfairness, not $T_{\mathcal{F}^{95}}$.

## VI. Related Work

Different definitions of and processes to reach reproducibility exist [20], e.g. as a three stage process as defined by an ACM policy [21]. Thereby, the minimum level is **repeatability**. It refers to recreating the results for an experiment conducted by the same scientists with the same tools. The term **replicability** is used for results that can be reproduced by other scientists given the same experiment setting. Finally, **reproducibility** defines that results can be validated in different experiments, by different scientists and tools.

Our paper contributes to these quality aspects by reproducing results of other scientists with different methods (i.e. *reproducibility*). By providing our framework as open source software, we increase the value of our results by allowing others to replicate them (i.e. *replicability*).

### A. Reproducible Measurements with Network Emulation

Handigol et al. [22] have shown that various network performance studies could be reproduced using Mininet. The Mininet authors published an editorial note [23] in 2017, wherein they describe efforts in reproducing research. They reproduced performance measurements of DCTCP, Multi-Path TCP (MPTCP), the TCP Opt-ack Attack, TCP Fast Open, and many more. Other research groups used Mininet in studies about TCP, such as the work from Paasch et al. [24], with a performance evaluation of MPTCP. Girardeau and Steele use Mininet in Google Cloud VMs to perform simple BBR measurements [25]. They use a patched kernel and, compared to our approach, their setup and runtime for one experiment is significantly higher with up to 50 minutes.

BBR support is announced to be available for the network simulator ns3 [26]. The Pantheon allows researchers to test congestion control algorithms in different network scenarios [27]. The results of Internet measurements are used to tune the parameters of emulated network paths which provides better reproducibility.

### B. TCP BBR in Other Domains

BBR deployed in domains with different requirements yields varying results. Kuhn has shown promising results over SATCOM links, which have latencies in the range of 500 ms [28]. They state that a “late-comer unfairness” [28] exists. Leong et al. claim that BBR can be further improved for mobile cellular networks [29], which is a recent research area of Cardwell et al. [5]. Kakhki et al. integrated BBR for QUIC, however, state that it is not yet performing well [30].

## VII. Conclusion

We presented a framework for TCP congestion control measurements focusing on flexibility, portability, reproducibility and automation. We used Mininet to emulate different user-configured flows. Experiments run without user interaction and produce a report containing graphs visualizing 14 metrics. We reproduced related work to validate the applicability of our approach using emulation.

Furthermore, we summarized the state of the art for analysis for TCP BBR and extend existing insights in several aspects. In particular, we have shown that the algorithm to determine the duration of the Probe RTT phase is flawed and that in most cases BBR and CUBIC do not share bandwidth in a fair manner. Our final contribution is an experimental analysis of the synchronization mechanism. We identified two primary problems. Depending on the timing of new flows joining existing flows in relation to their Probe RTT phase, bandwidth can be shared severely unfair. This boils down to BBR's general problem of overestimating the BDP. The second problem is the time until a bandwidth equilibrium is regained. This can last up to 30 s, which is bad for short-lived flows, common in today's Internet. We identified that this is correlated with the trigger for synchronization, i.e. the Probe RTT phase, draining the queues. Consequently, without reducing the time between Probe RTT phases, the worst case time until flows synchronize cannot be improved further.

Our framework as well as the raw data for all figures presented is available online [7] for replicability of our results and to allow further investigations by the research community.

## Acknowledgment

This work was supported by the High-Performance Center for Secure Networked Systems and the German BMBF project SENDATE-PLANETS (16KIS0472).

## References

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based Congestion Control," *ACM Queue*, vol. 14, no. 5, 2016.

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, 2008.

[3] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," in *Proceedings of the International Conference on Communications*, vol. 43, 1979.

[4] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *25th IEEE International Conference on Network Protocols (ICNP 2017)*, 2017.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, I. Swett, J. Iyengar, and V. Vasiliev, "BBR Congestion Control: IETF 100 Update: BBR in shallow buffers," *IETF 100*, 2017, Presentation Slides. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers/

[6] S. Ma, J. Jiang, W. Wang, and B. Li, "Towards RTT Fairness of Congestion-Based Congestion Control," *CoRR*, vol. abs/1706.09115, 2017. [Online]. Available: http://arxiv.org/abs/1706.09115

[7] Framework and Data Publication. [Online]. Available: https://gitlab.lrz.de/tcp-bbr

[8] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Commun. ACM*, vol. 55, no. 1, 2012.

[9] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," Tech. Rep., 2009.

[10] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, 1995.

[11] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats *et al.*, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015.

[12] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks*, 2017.

[13] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for high-speed and long Distance Networks," in *Proceedings-IEEE INFOCOM*, 2006.

[14] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the 2011 ACM SIGCOMM Conference*, vol. 41, no. 4. ACM, 2011.

[15] N. Cardwell, Y. Cheng, S. Yeganeh, and V. Jacobson, "BBR Congestion Control," Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, 2017. [Online]. Available: http://www.ietf.org/internet-drafts/draft-cardwell-iccrg-bbr-congestion-control-00.txt

[16] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[17] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," Internet Requests for Comments, RFC Editor, RFC 5166, 2008.

[18] R. Jain, D.-M. Chiu, and W. R. Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984, vol. 38.

[19] S. Ebrahimi-Taghizadeh, A. Helmy, and S. Gupta, "TCP vs. TCP: a systematic study of adverse impact of short-lived TCP flows on long-lived TCP flows," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2. IEEE, 2005.

[20] D. G. Feitelson, "From Repeatability to Reproducibility and Corroboration," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, 2015.

[21] O. Bonaventure, "April 2016: Editor's message," in *ACM SIGCOMM CCR*, 2016.

[22] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-based Emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. ACM, 2012.

[23] L. Yan and N. McKeown, "Learning Networking by Reproducing Research Results," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 2, 2017.

[24] C. Paasch, R. Khalili, and O. Bonaventure, "On the Benefits of Applying Experimental Design to Improve Multipath TCP," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. ACM, 2013.

[25] B. Girardeau and S. Steele. Reproducing Network Research (CS 244 '17): Congestion-based Congestion Control With BBR. [Online]. Available: https://reproducingnetworkresearch.wordpress.com/2017/06/05/cs-244-17-congestion-based-congestion-control-with-bbr/

[26] C. A. Grazia, N. Patriciello, M. Klapez, and M. Casoni, "A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control?" in *Proceedings of the 14th International Conference on Telecommunications (ConTEL)*. IEEE, 2017.

[27] Pantheon: The Training Ground for Internet Congestion Control Research. [Online]. Available: http://pantheon.stanford.edu

[28] N. Kuhn, "MPTCP and BBR performance over Internet satellite paths," *IETF 100*, 2017, Presentation Slides. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-mptcp-and-bbr-performance-over-satcom-links/

[29] W. K. Leong, Z. Wang, and B. Leong, "TCP Congestion Control Beyond Bandwidth-Delay Product for Mobile Cellular Networks," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017.

[30] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a Long Look at QUIC," in *Proceedings of the 2017 Internet Measurement Conference*, 2017.