Reproducible Measurements of TCP BBR Congestion Control

Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, Georg Carle

Chair of Network Architectures and Services, Technical University of Munich

{jaeger|scholz|raumer|fgeyer|carle}@net.in.tum.de

Abstract

The complexity of evaluating TCP congestion control has increased considerably since its initial development in the 1980s. Several congestion control algorithms following different approaches have been proposed to match the requirements of modern computer networks. We present a framework to analyze different congestion control algorithms using network emulation. The framework is publicly available which provides easy repeatability of our experiments and reproducibility of the results.

As a case study, we use our framework to analyze the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm, which was published by Google in 2016. Because of promising initial results, BBR has gained widespread attention. As such it has been subject analysis, which showed an increase in performance, but also revealed critical flaws. We verify our framework by reproducing experiments from related work which confirm weaknesses of the current BBR implementation. We also contribute an analysis of BBR's inter-flow synchronization behavior and its interaction when competing with other congestion control algorithms. Our results show that BBR flows on their own have difficulty to reach a fairness equilibrium and suppress other congestion control algorithms. BBR is still work in progress, thus the framework is useful to validate further updates of BBR by rerunning the given experiments.

Keywords: TCP, Congestion Control, BBR, Reproducible Measurements

1. Introduction

TCP congestion control is imperative for applications to utilize network resources efficiently. Over the years, several algorithms have been developed with different characteristics [1]. This can be algorithms adapted to a specific domain, or different metrics used to calculate the congestion window. This results in multiple congestion control algorithms being deployed in the Internet at the same time [2], respective flows competing for available bandwidth based on different congestion indicators. To analyze the efficiency of an algorithm it is therefore not enough to just study a single mechanism, but also the interaction when competing with other congestion control algorithms.

We introduce a framework for automated and reproducible measurements of TCP congestion control algorithms. It uses Mininet as backend for emulating network topologies with Linux network namespaces. The flexible configuration enables manifold scenarios, whereby important values and metrics are extracted and post-processed automatically. This allows to analyze the behavior of different TCP congestion control algorithms and their interaction with each other in detail. It produces repeatable experiments and is available as open source at [3]. The use of emulation using Mininet allows the framework to be independent of hardware constraints, 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 different congestion control algorithms in various 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 regarding inter-flow unfairness and inter-protocol fairness when different algorithms compete with each other.

As a case study we use the framework to analyze TCP BBR, a congestion-based congestion control algorithm developed by Google and published in late 2016 [4]. In contrast to traditional algorithms like CUBIC [5] 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 [6] 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 [4]. Google already deployed BBR in its own production platforms like the B4 wide-area network and YouTube to develop and evaluate BBR [4] 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. [7, 8, 9]). As a consequence, BBR is actively improved [8]. Proposed changes usually aim to mitigate specific issues, however they need to be carefully studied for unintended side effects.

We deepen the analysis of BBR regarding inter-flow unfairness and inter-protocol fairness when competing with TCP CUBIC, Reno, Vegas and Illinois flows. Lastly, we use measurements to analyze the inter-flow synchronization behavior of BBR flows.

This paper is structured as follows: Section 2 presents background to TCP congestion control. In Section 3, we describe our framework for reproducible TCP congestion control 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 4. Section 5 demonstrates the interactions when BBR flows compete with other, loss-, delay- and loss-delay-based congestion control algorithms. Our BBR inter-flow synchronization analysis is discussed in Section 6. Related work is presented in Section 7 before we conclude with Section 8.

2. 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 [10]: poor queue management and failure of TCP congestion control. Algorithms like TCP NewReno [11] or TCP CUBIC [5] 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.

Various TCP congestion control algorithms were developed to improve on loss-based congestion control. Examples include TCP Vegas [12], adapting delay as indicator, or TIMELY [13] based on precise RTT measurements. However, these are suppressed when competing with loss-based algorithms. Hock et al. present TCP LoLa [14], primarily focusing on low latency. Hybrid algorithms using both loss and delay as congestion indication were proposed such as TCP Compound [15] or TCP Illinois [16]. Alizadeh et al. proposed Data Center TCP (DCTCP) [17], which requires support for Explicit Congestion Notification (ECN) in network switches. Utility functions were also proposed in order to describe objectives for congestion control, such as the works from Winstein and Balakrishnan in [18] with TCP Remy, or the work from Dong et al. in [19] with Performance-oriented Congestion Control (PCC).

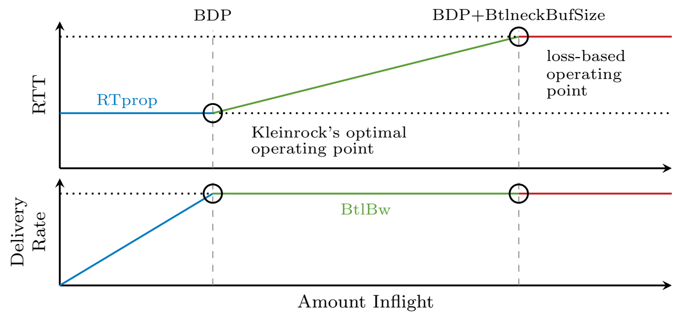


Figure 1: Effect of increasing inflight data on the RTT and delivery rate. Based on [4].

2.1. 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 BtlBw $\cdot$ RTprop describes the amount of data that can be inflight (non-acknowledged) to fully utilize the network path, coined Kleinrock's optimal point of operation [6].

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 BDP + 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.

2.2. Loss-based Congestion Control

A simple idea to detect congestion is to assume that each packet loss solely happens due to congestion. Algorithms following this approach are classified as loss-based congestion control algorithms. Popular loss-based TCP versions are Reno [11], BIC [20] and CUBIC [5]. They only use a congestion window as control parameter which limits the amount of unacknowledged data in the network. Its size grows as long as all packets arrive and is reduced whenever packet loss occurs. The quantity of theses increases and decreases usually varies for different loss-based algorithms For example, Reno increases its congestion window by one for each RTT and reduces it by 50 % whenever packet loss is detected. As a result, Reno has both problems with RTT fairness and the utilization of long-delay links with large BDP.

TCP CUBIC increases the window according to a cubic slope as a function of the time since the last packet loss happened. It sets an inflection point as target, the area where it estimates to reach the optimum operating point. Close to the inflection point, CUBIC is conservative, trying to have a stable congestion window, however, the further away CUBIC gets from the inflection point, the more aggressive the congestion window is increased. In case of a loss event, CUBIC adjusts its estimated target conservatively reducing it by 20 % [1].

Loss-based approaches suffer from two major problems. First, they are susceptible to random packet loss since TCP interprets them as a signal for congestion and steadily reduces its congestion window, leading to under-utilization of the link. Secondly, they shift their operation point away from Kleinrock's optimal operation point (Figure 1). This permanently keeps the network buffers full, which is not a problem as long as the buffer size is small. However, with larger size the additional buffer delay grows increasing the transmission delay of all connections using that link. This becomes a major problem for real-time applications like telephony or streaming via TCP.

2.3. Delay-based Congestion Control

Delay-based algorithms use the measured time between a packet was sent and the corresponding acknowledgement arrived to detect congestion. If this time increases, TCP assumes that a queue has formed somewhere on the path and reduces the sending rate. Thus, compared to the loss-based approach, congestion can be detected before any packet-loss occurs.

TCP Vegas [12] is an example for a delay-based algorithm. It periodically measures the connection's RTT and stores the minimum measured as RTT_{min}. The congestion window is scaled linearly according to the difference between the measured RTT and RTT_{min}. Since Vegas keeps RTT_{min} for the whole connection it cannot adapt to changes in the path's RTprop.

Delay-based congestion control algorithms are known to perform poorly when run in parallel with loss-based algorithms [21].

2.4. Hybrid Congestion Control

To combine the advantages of both approaches, hybrid algorithms were developed. TCP Illinois [16] uses packet loss as the primary signal for congestion, which decides if the congestion window should be increased or decreased. Additionally, the delay is taken into account to determine the quantity of the change. When the network is not congested, Illinois grows the congestion window fast and reduces it less drastically. When delay increases the growth is slowed down, leading to a concave slope similar to CUBIC. Another example for a hybrid algorithm is Compound TCP [15], having both a delay-based and a loss-based component.

2.5. Bottleneck Bandwidth and Round-trip Propagation Time

Cardwell et al. proposed TCP BBR following a new approach named congestion-based congestion control, which is supposed to react only to actual congestion and not only to indicators as former algorithms. In this section the basics of BBR that are important for our evaluation are described. Our deliberations are based on the version presented by Cardwell et al. [4] and we refer to their work for a detailed description of the congestion control algorithm or [7] for a formal analysis.

2.5.1. 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.

2.5.2. Phases

The BBR algorithm has four different phases [22]: 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

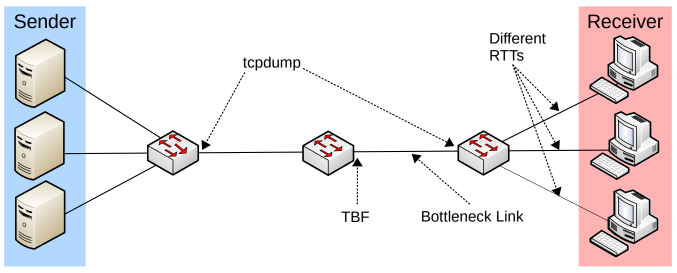


Figure 2: Mininet setup with sending and receiving hosts and bottleneck link.

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.

3. 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 without the need for high performance hardware and testbed. The complete measurement process shall be simplified through **Automation**. Via configuration files and experiment description, including post processing of data and generation of plots, the experiment should be executed without further user interaction.

The full source code of our framework is available online [3].

3.1. 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 [23]. 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 4 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.

3.2. Setup

Topology: As a TCP connection can be reduced to the bottleneck link (cf. Section 2.1), 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 & Packet Loss: 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 [7], but works well for the data rates we use. Additionally, a stochastic packet-loss rate can be specified.

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 [4].

BBR Pacing: To send data at the desired rate, BBR requires the TCP stack to pace outgoing packets. For Linux versions before 4.13, pacing support was not part of the TCP stack but implemented by the Fair Queue (FQ) queuing discipline. Since we performed our measurements on Linux 4.9, we explicitly configured the FQ queuing discipline on each BBR sender, but we verified that using FQ is no longer required in Linux 4.13.

3.3. 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. During the execution of the test only data is collected and nothing is analyzed yet. Thus, no collected data is altered due to computational processes running on the same system and the collection and analysis can be split up to different systems and moments in time. The analysis framework then automatically extracts data and computes the implemented metrics. This automation allows to systematically evaluate parameter spaces. For example, Figure 12 shows the results of more than 800 individual experiments.

The analysis framework outputs CSV files for each computed metric to simplify further processing of the data. Additionally a plot visualizing all gathered data is generated in form of a PDF file.

3.4. 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 in customizable intervals. We capture the packet headers up to the TCP layer of all packets before and after the bottleneck using `tcpdump`.

The raw data is processed afterwards 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. It uses the `dpkt`[1] module to extract the information from the packet captures.

As a result of one experiment, a report containing 14 graphs visualizing the metrics over time is automatically generated. Sample configuration files can be found with our source code publication [3].

Sending Rate & Throughput: A useful metric for analyzing TCP fairness is the sending rate, which can be larger than the maximum transmission rate of the link, resulting in congestion at the bottleneck. We compute the per flow and total aggregated sending rate as the average bit-rate based on the IP packet size in modifiable intervals, using the capture before the bottleneck. The time interval $\triangle t$ should be small enough to deliver precise values and large enough to avoid oscillations. E.g. if $\triangle t$ is so small that either 1 or 0 packets arrive per interval, then the computed sending rate alternates between 0 and $\frac{\text{packet size}}{\triangle t}$.

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 [24] and use Jain's Index [25] 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: We use the TCP Timestamp option [26] to measure the current RTT for each arriving acknowledgment. To reduce the amount of data points, the average RTT is computed in time intervals.

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.

[1]`https://pypi.python.org/pypi/dpkt`

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.

Bottleneck Buffer Backlog: We use Linux' traffic control tool to extract the current backlog of the bottleneck buffer. We count the backlog length in bits in a modifiable interval, e.g. 20 ms. This interval was chosen to see the impact of short-lived effects without adding too much computational and storage overhead. This allows to get a clearer view of the current congestion of the bottleneck.

Congestion Control Internal Values: Congestion control algorithms keep different internal metrics, like for instance the congestion window or the slow start threshold. In addition, 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.

3.5. Limitations

Due to resource restriction on a single host emulated network, the framework is limited both by the bottleneck bandwidth and the number of concurrent flows. We ran tests both on a normal notebook and on more powerful commercial off-the-shelf server hardware with different CPUs. In Figure 3 the CPU utilization of different host systems running the framework is depicted. The used bandwidth hardly impacts the performance during the emulation step, but increases the duration of the analysis step since the packet capture files grow in size. The number of flows is limited by the CPU since we use individual sender and receiver hosts for each flow, each having its own network namespace. To provide valid results the host system's resources should not be fully utilized. For example, we observed a drop in the polling rate of the internal TCP values at about 50 % CPU utilization for each test. Though, the results show that when using a bandwidth below 10 Mbit/s and up to 20 concurrent flows, 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.

4. 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

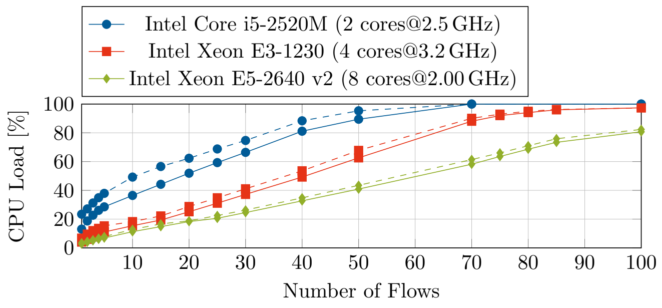


Figure 3: CPU Utilization when running the framework using 10 Mbit/s (solid) and 100 Mbit/s (dashed) links.

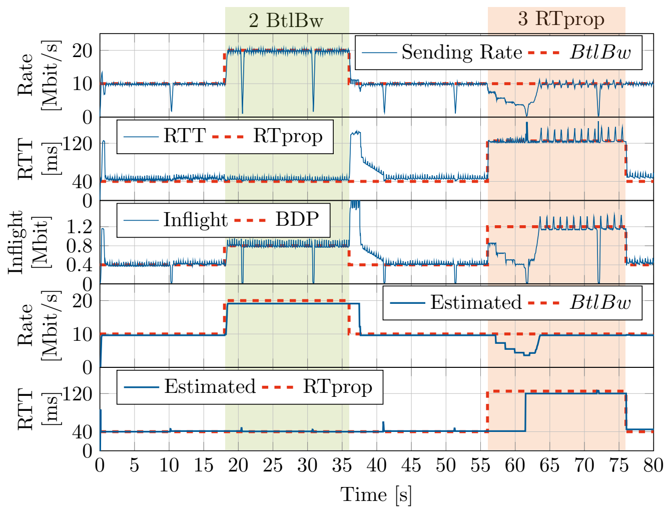


Figure 4: Single BBR flow (40 ms, 10 Mbit/s bottleneck) under changing network conditions. Values sampled every 40 ms. The bottom two plots show BBR's internal metrics.

their current research efforts towards BBR 2.0 [4, 8]. 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 [7]. 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 [3]. Unless representing a single flow, measurements were repeated five times and standard deviations are shown where applicable.

4.1. Single Flow

Figure 4 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 [4, 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, the internal BtlBw estimation adapts it few seconds later so a queue is generated, as indicated by the increased RTT, 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, which BBR sets to 2 BDP. To adapt, BBR has to limit 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. However, larger increases of the RTT can lead to BBR utilizing less than 20 % of the available bandwidth for up to 10 s. Second, it is unlikely that such a drastic change in RTT happens in the Internet in the first place.

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

Figure 4 also validates that our framework can sample events detailed enough ($\triangle t = 40$ ms), as both Probe Bandwidth (small spikes) and Probe RTT phases (large spikes every 10 s) are displayed accurately. However, in general we use $\triangle t = 200$ ms for less overhead.

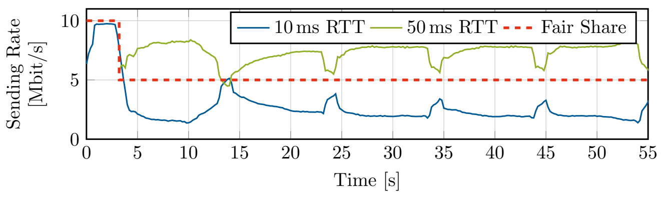
4.2. RTT Unfairness

The RTT unfairness of BBR is visualized in [9, 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 lower RTT starts three seconds before the other. We set the bandwidth to 10 Mbit/s and adapted all other parameters. Our reproduced results (Figure 5a) only differ slightly: The larger flow receives about 10 % less of the bandwidth.

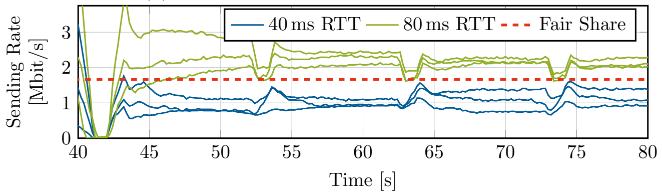
As shown in Figure 5b, the behavior can also be observed when increasing the number of flows. Flows with equal RTT converge to a fair share within their group, however, groups with higher RTT claim a bigger share overall.

4.3. Bottleneck Overestimation for Multiple Flows

BBR overestimates the bottleneck when competing with other flows, operating at the inflight data cap [7]. The analysis 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 6. For five simultaneously started BBR flows, the sum of the BBR estimations of BtlBw exceeds

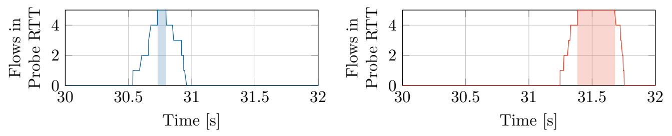


(a) Two BBR flows with different RTTs

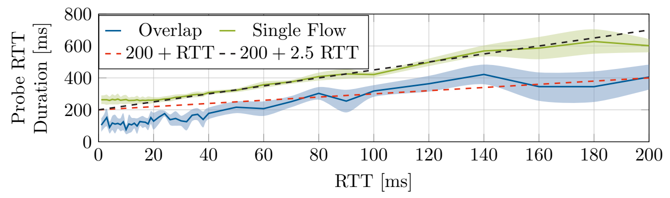


(b) Multiple BBR flows with two groups of RTTs

Figure 5: RTT Unfairness of BBR



(a) $RTprop = 10\,\text{ms}$ (b) $RTprop = 100\,\text{ms}$



(c) Overlapping Probe RTT phase duration

Figure 7: Influence of different RTT on simultaneous Probe RTT phases

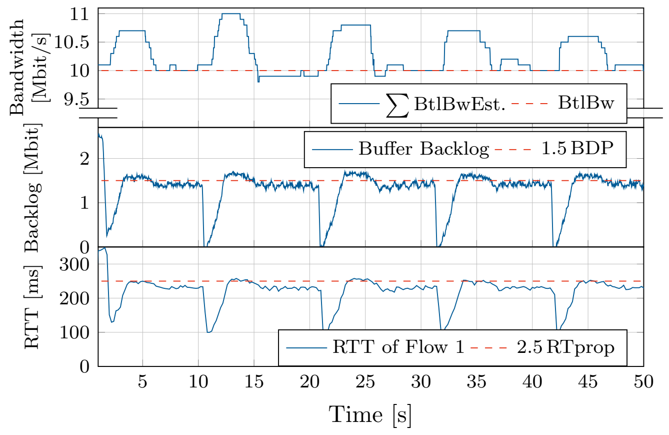


Figure 6: BDP overestimation for five flows with a 100 ms RTprop and 10 Mbit/s bottleneck (5 BDP buffer)

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.

4.3.1. Insufficient Draining of Queues During Probe RTT

To measure the correct RTprop value, all flows need to simultaneously drain the queue in Probe RTT. Otherwise, the queue cannot be drained since the BBR flows not being in Probe RTT measure an increased BtlBw resulting in a higher sending rate. This prevents all BBR flows from measuring a low RTprop value. Figures 7a and 7b show the overlap of the Probe RTT phase for five synchronized flows for different RTprop values. While all flows eventually are in the Probe RTT phase simultaneously, they arrive with slight offsets as no perfect inter-flow synchronization can be achieved. As a consequence, the duration for which all flows are draining the queue (during Probe RTT) is only a fraction of the actual duration of the Probe RTT phase. In this case the last flows join the Probe RTT phase while the first flows are already leaving again. Consequently, the queue is not drained completely, resulting in an overestimation of the bottleneck. This becomes more severe for shorter estimated RTTs.

The duration of Probe RTT increases for a wider range of different RTTs. Figure 7c displays the duration of the Probe RTT phase for five flows, and the resulting overlap. 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 [22]. 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.

4.3.2. 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 [8].

We reproduced this effect by analyzing the relation between bottleneck buffer size and caused retransmissions for both BBR and CUBIC (cf. Figure 8). Five TCP flows are started simultaneous and share a 10 Mbit/s, 50 ms bottleneck. We compute the retransmission rate for different buffer sizes at the bottleneck for BBR and CUBIC individually. Since BBR overestimates the bottleneck during startup and synchronization (cf. Section 6), we focused on the retransmission rates during normal steady state behavior. Thus, we measured the rates about 25 s after the flows started.

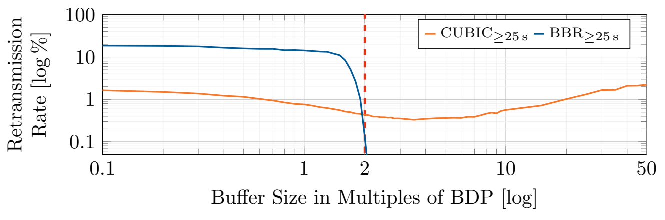


Figure 8: Retransmission rate after 25 s for 5 simultaneously started flows with different bottleneck buffer sizes

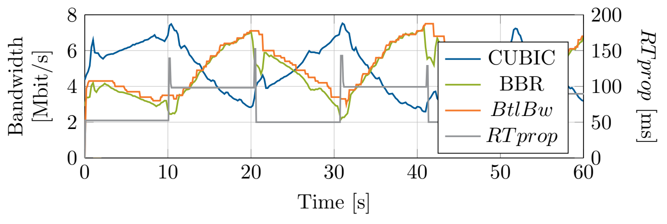


Figure 9: Competing BBR and CUBIC flow

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 constant bottleneck overestimation, in contrast to CUBIC's adaption of the congestion window for loss events. For larger buffers BBR can operate without a single packet 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 [10] and therefore only pose a theoretic problem.

5. Inter-protocol Behavior

Our emulation framework creates a new sender/receiver host pair for each flow. This allows to configure different congestion control algorithms per flow and analyze the interaction thereof. The following section presents our case study on BBR's inter-protocol behavior, when competing with either the loss-based CUBIC and Reno, the delay-based Vegas or the loss-delay-based Illinois congestion control algorithms.

Liu et al. define requirements for congestion control algorithms [16]. Amongst others, two requirements are important if a newly proposed algorithm should be usable in the Internet. First, the new algorithm should not perform significantly worse when competing against any other algorithm which is used in the Internet. Otherwise, there is no incentive for anyone to use the algorithm at all. Second, other algorithms should not receive major throughput reductions, i.e., the bandwidth should be shared in a fair manner between different algorithms.

To accomplish these two goals BBR should neither be too aggressive nor too gentle towards other algorithms. We focus our comparison mainly on TCP CUBIC, as it is the current default congestion control algorithm in the Linux kernel.

5.1. Competing with Loss-based Algorithm

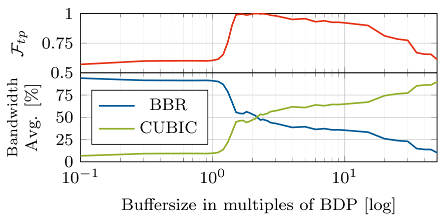
In this section BBR's fairness towards loss-based congestion control algorithms such as Reno or CUBIC is evaluated.

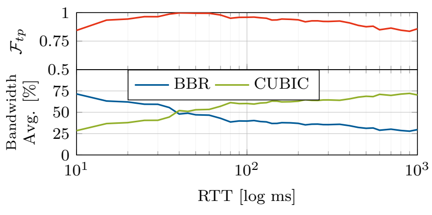
Influence of Changing Network Parameter: In the best case, regarding fairness, a competing BBR and CUBIC flow reach an oscillating steady-state [8]. This is caused by the RTprop estimation of BBR as shown in Figure 9. CUBIC's aggressive probing for bandwidth causes the queues to fill up, resulting in BBR to measure a higher delay, increasing its BDP. In turn, this causes packet loss, resulting in reduced inflight data 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 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 to reduce the impact of these oscillations.

The size of the bottleneck buffer is crucial for the fairness between competing BBR and CUBIC flows [4, 7]. Figure 10a 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 4.3.2).

While showing the same overall behavior, RTT changes have a smaller influence on the fairness if applied to both flows as shown in Figure 10b. 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 10c). 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 a growing share with increasing RTT. Our conclusion is that the fairness between CUBIC and BBR largely depends on the bottleneck buffer size, while the RTT only has a small impact.

Increasing Number of Competing Flows: Lastly, we evaluate how the number of flows competing with each


(a) Increasing buffer with 50 ms RTT


(b) Increasing RTT with 2.3 BDP buffer

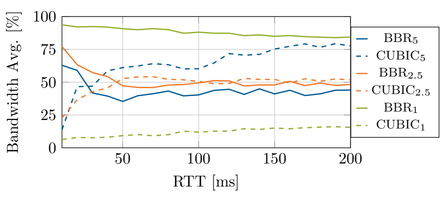

(c) One flow 50 ms RTT for 1, 2.5 and 5 BDP buffer

Figure 10: One CUBIC vs. one BBR flow for changing network conditions

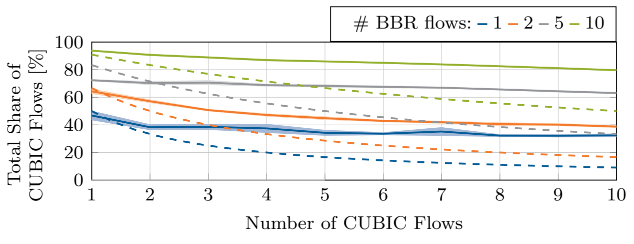


Figure 11: Bandwidth share of different number of CUBIC and BBR flows competing. Dashed lines show fair share.

other influences the throughput share per congestion avoidance algorithm.

Figure 11 shows that CUBIC is suppressed independent of the number of flows in a scenario with 50 ms RTT and 2.5 BDP bottleneck buffer, the bottleneck configuration with highest $\mathcal{F}_{tp}$ (cf. Figure 10). 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.

Beside the absolute values of the bandwidth share BBR claims, we also analyzed how far the share deviates from their actual fair share. For this purpose, we run measurements with one to ten flows of different congestion control algorithms against one to ten BBR flows for 240 s on a 50 ms, 10 Mbit/s bottleneck with a buffer size of 2.5 BDP. We compute the difference between the average throughput share BBR achieves for each test and the actual fair share. The results are visualized in Figure 12. Positive values mean that BBR is unfair towards the other algorithm. This is the case for most of the test while BBR never falls more than 10 % below its fair share.

For example, when running against CUBIC, BBR gets a larger share for all tests with more than two CUBIC flows independent of the number of BBR flows (see Figure 12b). Furthermore, an increase of the CUBIC flows enhances this unfairness and for more than five flows, CUBIC gets at least 20 % less than its fair share. The reason for this is that more CUBIC flows fill the buffer faster generating packet loss more often. This makes the CUBIC flows back off and allows BBR to claim a larger bandwidth share. Additionally, more loss-based flows prevent BBR from draining the queue during Probe RTT resulting in an overestimation of RTprop which leads to an even larger share for BBR. Only two or less CUBIC flows can manage to get more bandwidth than their fair share at all.

There is a visible optimum for CUBIC when running with one flow against five or six BBR flows. For this, we found the two following reasons.

First, an increasing number of BBR flows allows it to drain the whole queue during Probe RTT and thus all BBR flows have lower RTprop estimations. If the queue is not completely drained, the RTprop estimation usually oscillates between the actual RTprop and a larger overestimation (cf. Figure 9). Thus, when BBR can measure a low RTprop, the estimated BDP also decreases and it keeps less data inflight leaving more space for CUBIC. Figure 13 shows the distribution of RTprop estimations with increasing number of BBR flows competing against one CUBIC flow. The overestimations decrease while more BBR flows are running since more flows drain the queue simultaneously. Actually, more than four BBR flows can empty the queue for most of the Probe RTT phases, which allows measuring low values for RTprop.

Secondly, more BBR flows require a larger buffer to hold the persistent queue and run without any deterministic loss. Although, we use the optimal buffer size of 2.5 BDP from Figure 10a in the experiments, however, more flows create even larger queues (Figure 14). This results in BBR failing to completely drain the queue since the time when all flows are in Probe RTT is too short for the given RTT of 50 ms. For more than seven BBR flows, this persistent queue is even larger than the used buffer size, which leads to constant packet loss caused by BBR. Furthermore, since BBR does not react to this implicit signal for congestion, it has higher retransmission rates than CUBIC. When running each ten BBR and CUBIC flows, 80 to 90 % of all retransmissions are sent by BBR.

These two reasons result in CUBIC getting about 10 % more bandwidth than its fair share with one flow against five BBR flows, which is the optimum for CUBIC (see Figure 12b).

We also compared BBR to TCP Reno. Competing with BBR, Reno shows similar results as CUBIC, since both are loss-based algorithms (see Figure 12a). However, CUBIC improved many of Reno's weaknesses and thus the overall performance of Reno against BBR is worse.

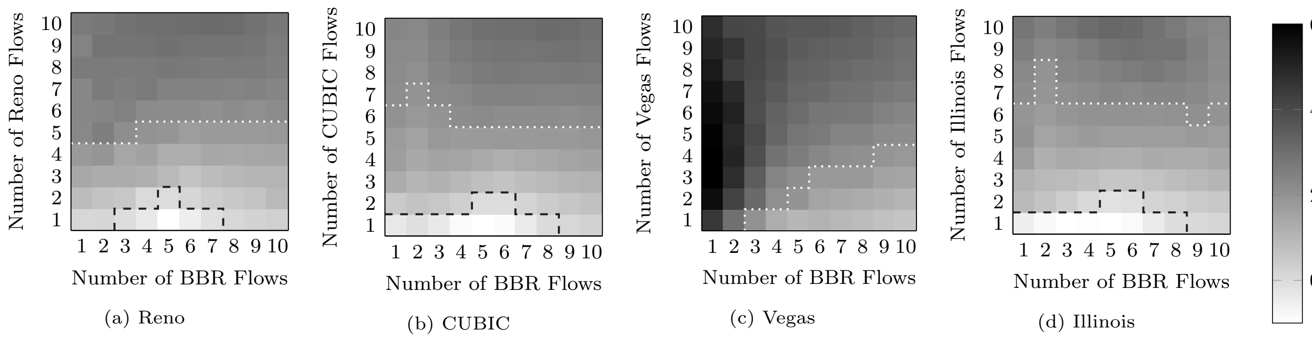


Figure 12: BBR's fairness towards other algorithms. The shading describes the difference between the actual gained share of all BBR flows and the fair share in percent. Positive values indicate that BBR is claiming more bandwidth than it should. Contour lines are given for 0 (dashed) and 20 % (dotted).

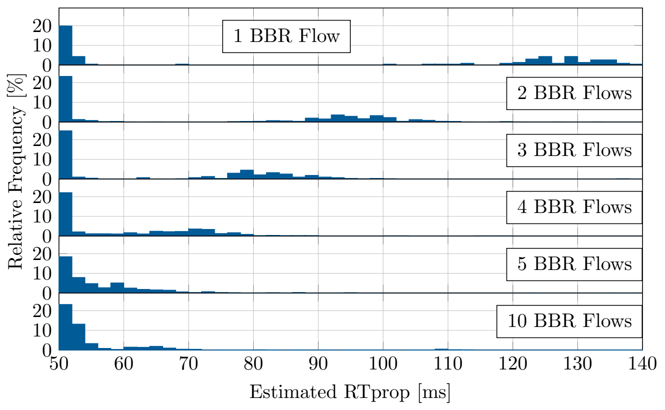


Figure 13: Relative frequency of RTprop estimations for different number of BBR flows competing with one CUBIC flow on a 50 ms link.

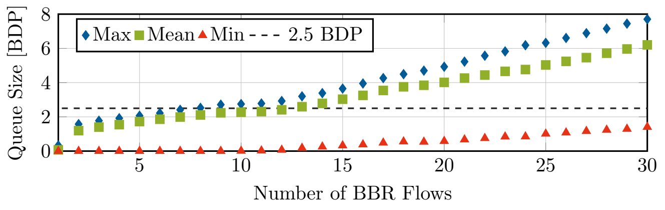


Figure 14: Size of the persistent queue at the bottleneck buffer with increasing number of BBR flows. More than ten BBR flows fail to drain the queue completely.

5.2. Competing with Delay-based Algorithm

Similar to BBR, Vegas also tries to keep the delay as low as possible. Therefore, not only the fairness between Vegas and BBR is evaluated, but also whether both algorithms can keep the bottleneck queue small when running simultaneously.

Regarding the throughput, Vegas performs poorly when running parallel with BBR. Independently of the order in which the flows are started, Vegas receives only 5 to 10 % of the total bandwidth (see Figure 15). After the BBR flow starts, a queue is created at the bottleneck since Vegas already fully utilizes the link. This induces additional delay and Vegas reacts accordingly by decreasing its congestion window. Hence, BBR can measure higher values for BtlBw increasing its sending rate further until Vegas' congestion window cannot get any lower.

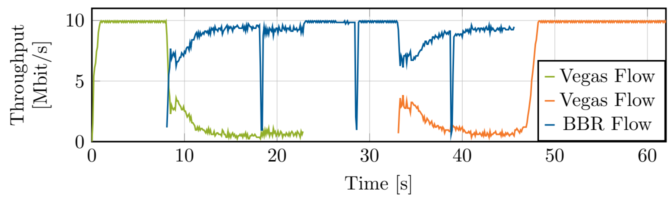


Figure 15: BBR competing with Vegas

In this experiment, BBR makes accurate RTprop and BtlBw estimates. Still, it does not reach its desired operation point and is capped by its congestion window, which adds one BDP of data to the queue. When the BBR flow is in Probe RTT, the Vegas flow measures a lower RTT and starts increasing its congestion window. After Probe RTT, BBR returns to its previous sending rate. The total sending rate of both flows exceeds the link's bandwidth and a queue is created. Since Vegas cannot decrease its congestion window further and BBR only adjusts its sending rate to its BtlBw estimation, this queue cannot be removed until the next Probe RTT phase. Then the whole process repeats. Thus, a parallel Vegas flow does hardly impact the throughput of BBR but it manages to push BBR into a state in which BBR maintains a persistent queue without the capability of draining it.

When considering throughput, Vegas performs poorly against BBR independently of the number of Vegas flows (Figure 12c).

5.3. Competing with Loss-delay-based Algorithm

Lastly, we compare BBR to the loss-delay-based Illinois algorithm.

Figure 12d is similar to the other loss-based algorithms since Illinois still uses packet loss as primary signal for congestion. However, Illinois can achieve higher throughput against BBR than Reno, because Illinois slows down when increasing delay is measured, which also increases

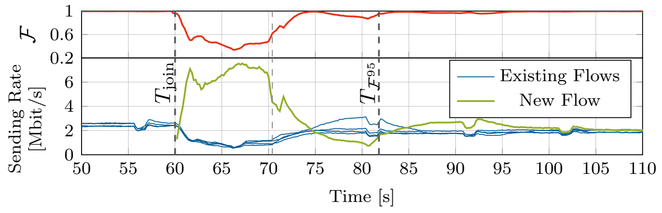
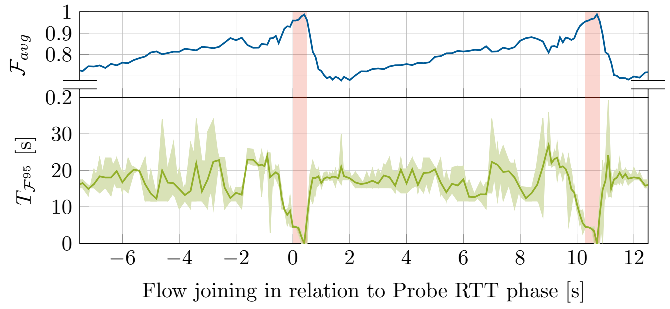
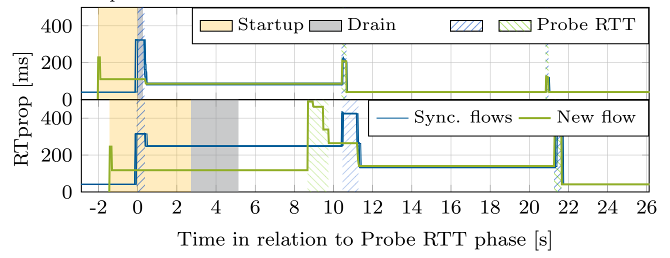


Figure 16: BBR inter-flow synchronization behavior



(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 17: Single BBR flow joining synchronized BBR flows

the time until the next congestion event occurs. Additionally, Illinois is affected less by the constant packet loss with increasing number of BBR flows due to the smaller multiplicative decrease.

5.4. Summary

Overall, there are several different factors, as buffer size, RTT or number of flows which influence the fairness of BBR towards other algorithms. For competing loss-based algorithms, the deciding factor is the used buffer size while Vegas completely starves but manages to move BBR's operation point towards a persistent queue at the bottleneck. For most configurations, BBR received a too large share, mostly due to being too aggressive and not considering any implicit congestion signals. However, BBR did not starve against another algorithm in any of our tests.

6. 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.

6.1. Theory & Questions

Cardwell et al. demonstrate in [4, Fig. 6] how different BBR flows synchronize whenever a large flow enters the Probe RTT phase. We visualize the process in Figure 16 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 [4, 22]. 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 adding new flows can result in extended or accumulated unfairness.

6.2. 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_{join} as the point in time when the flow of interest, i.e. the last flow, has joined the network (cf. Figure 16). As first metric, we define $T_{\mathcal{F}^{95}}$ as the point after T_{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 4.2 with RTT unfairness between flows.

6.3. 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 17a shows our experimental evaluation when joining a new flow in relation to the Probe RTT phase of the synchronized flows.

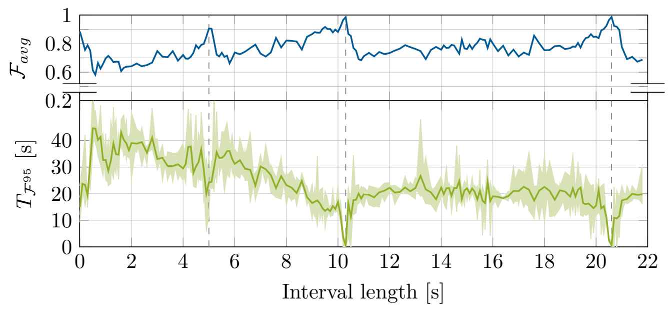


Figure 18: Different join intervals for subsequent flows

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 $T_{\mathcal{F}^{95}} = 10$ s is because the existing flows enter the Probe RTT phase while the new flow drains after the Startup as shown in Figure 17b. 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 $T_{\mathcal{F}^{95}} = 22$ 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 [27, 5], 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.

6.4. 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 18).

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

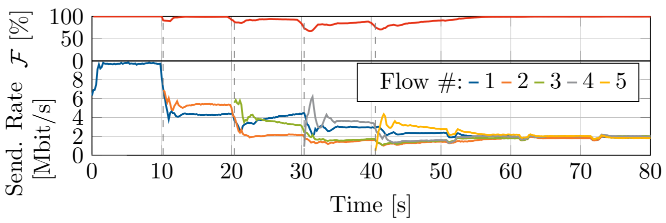
Analyzing the effect of a new flow joining on individual existing flows, e.g. the longest running flow, is difficult for the lack of a good metric. We therefore select the best and worst case join intervals displayed in Figure 19 for a visual analysis. As the minimum value of $\mathcal{F}$ depends on the number of flows ($1/n$), it is normalized using percentages.

Figures 19a and 19b 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.

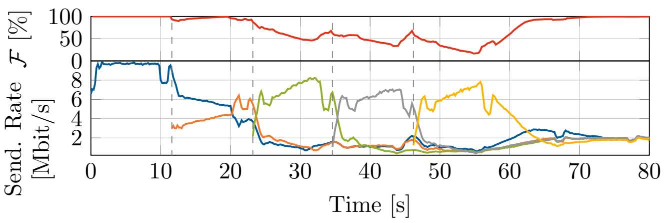
6.5. Further Observations

Further measurements showed that many BBR flows also have problems to synchronize. When running 50 BBR flows, the required buffer size to run without packet-loss is about 10 BDP on a 10 Mbit/s, 100 ms link. If the buffer is too small, BBR constantly overwhelms the network and completely fails to synchronize. In this case no more than 10 % of the flows are in Probe RTT at the same time. Otherwise, the flows can synchronize and reach a fair bandwidth share. However, two issues are remaining. First, if the RTT of the flows is too short not all flows are in Probe RTT at the same time. This leads to undrained queues and RTprop overestimations (cf. Figure 7). Second, the whole synchronization process takes more than one minute when starting the flows in intervals of 0.2 s.

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 $T_{\mathcal{F}^{95}}$.



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



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

Figure 19: Identified best/worst case join intervals

7. Related Work

7.1. Quality of Scientific Experiments

Quality of scientific experiments includes aspects like objectivity, validity, reliability, repeatability, provability and consistency. However, not all aspects defined are of relevance for scientific experiments in general. E.g. objectivity is not an issue for a machine counting packets per second.

To judge and demonstrate the quality and validity of scientific experiments, reproduction through independent scientists is required. For this, not only experiment results have to be published, but also access to meta-data, including configuration files, tools, and scripts used throughout the experiment, has to be granted. Reliability, meaning overall consistency, of measurements is a continuous concern in the scientific computer networks community [33, 34]. It characterizes if a result can be reproduced by repetition of the test. Reliability has different facets whereof the usage differs with the field of science and is even within a field not used consistently [33].

Different definitions of and processes to reach reproducibility exist [33], e.g. as a three stage process as defined by an ACM policy [34]. Thereby, the minimum requirement 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. To allow replicability, all measurement data, tools, scripts and artifacts of the experiment have to be made accessible. Finally, **reproducibility** defines that results have to be validated in different experiments, by different scientists and tools, requiring significant time and resource investment. [33, 35, 36]

Although reproducibility is acknowledged as valuable in research, it is mostly not provided in publications. As identified by Bajpai et al. [37] this is a consequence of a missing incentive to perform the additional efforts that are required to achieve reproducibility. A way out is suggested by Scheitle et al. [38] who demand to include checks and rewards for reproducible research in the review process to provide incentives.

Our group has expressed dedication to foster reproducible research in computer networking [35, 38]. This not only includes proposing a new ecosystem that incentivizes reproducible research [38], but also adjusting methodologies to implement workflows orchestrating reproducible measurements [35].

Our paper contributes to these quality aspects by reproducing results of other scientists with different methods (i.e. *reproducibility*) as shown in Table 1. By providing our framework as open source software, we increase the value of our results by allowing others to replicate them (i.e. *replicability*). This also includes configuration files and scripts to repeat all measurements and figures presented in this work [3].

7.2. Reproducible Measurements with Network Emulation

Our framework relies on Mininet as a basis for evaluating the performance of TCP. Handigol et al. [39] have shown that various network performance studies could be reproduced using Mininet. The Mininet authors published an editorial note [40] 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. [41], with a performance evaluation of MPTCP.Girardeau and Steele use Mininet in Google Cloud VMs to perform simple BBR measurements [42]. 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 [43]. The Pantheon allows researchers to test congestion control algorithms in different network scenarios [44]. The results of Internet measurements are used to tune the parameters of emulated network paths which provides better reproducibility.

7.3. 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. [8]. Li et al. have compared TCP CUBIC and BBR “under highway driving conditions” [31]. Their results show that BBR

Table 1: State of TCP BBR

	Related Work	Our Contribution	Notes
Validation and Extended Insights			
Single flow behavior	[4]	Section 4.1, Reprod. Fig. 4	-
Adaption to RTprop increase		Section 4.1	Slow adaption
RTT unfairness	[7, 8, 9]	Section 4.2, Reprod. Fig. 5a	BBQ [9], Work-In-Progress [8]
Multi-flow bottleneck overestimation	[7, 8]	Section 4.3, Reprod. Fig. 6	BBR 2.0: "drain to target" [8]
Insufficient queue draining	[7, 8]	Section 4.3.1	Probe RTT not overlapping
Shallow buffer loss rate	[7, 8]	Section 4.3.2	BBR 2.0: "full pipe+buffer" [8]
New Aspects			
Inter-protocol suppression: CUBIC, Reno, Vegas, Illinois	[7, 8]	Section 5	BRR suppresses delay-, loss- and loss-delay-based algorithms
Inter-flow synchronization	[4]	Section 6	Best/Worst case analysis
BBR in Other Domains			
SATCOM	[28]	7.3	"Late-comer unfairness" [28]
Mobile Cellular Networks	[29, 30]	7.3	Improvements planned [8]
BBR over LTE (driving on highway)	[31, 30]	7.3	Comparison with CUBIC
Integration with QUIC	[32]	7.3	Planned [32]

achieves similar throughput with decreased average delay compared to CUBIC for large files, but higher throughput with higher self-inflicted delay for small downloads [31]. Atxutegi et al. also tested BBR on cellular networks, resulting in BBR performing better than other algorithms such as CUBIC and Reno [30]. However, they also detected that BBR has problems when encountering long or 4G latencies. Crichigno et al. measured an improved performace of BBR flows with larger maximum segment sizes using parallel streams [45]. This especially effects long living flows transporting huge amounts of data, so called elephant flows. Integration of BBR for QUIC is work in progress [32].

7.4. Further Development of BBR

Since its first publication, BBR has been under active development by the authors and research community. At IETF 102 Cardwell et al. proposed ideas for improving on several of the discovered issues [46], calling the algorithm BBR2.0. The slow synchronization is addressed by reducing the time span between the Probe RTT phases for example to 2 s. This is supposed to improve the fairness towards other BBR and also CUBIC flows. Still, it remains a challenging task to reach fairness with loss-based flows. Other areas of research are how BBR can handle fluctuating RTTs, as in WiFi environments or in the presence of delayed acknowledgements, and the influence of ACK aggregation [47]. Furthermore, explicit congestion notifications (ECN) and also packet-loss are taken into account to improve BBR's network model. This reduces the retransmission rate of BBR when running on shallow buffers. Lastly, the problem of BBR maintaining a persistent queue when running parallel with other BBR flows is addressed by adding mechanisms to drain existing queues more frequently.

Once these improvements are included in the Linux kernel, our framework can easily be used for validation by rerunning all tests and see how the results have changed. For all tests in this paper the scripts to generate the configurations and analyze the results are published at [3].

8. Conclusion

We presented a framework for TCP congestion control measurements focusing on flexibility, portability, reproducibility and automation. Using Mininet to emulate different user-configured flows, it allows to perform experiments analyzing a concrete algorithm, or the interaction with multiple flows using even different algorithms. We reproduced related work to validate the applicability of our approach.

We use the new TCP BBR algorithm as case study for our framework, summarizing the current state of the algorithm and extending existing insights in several aspects. In particular, we have shown that the algorithm to determine the duration of the Probe RTT phase has problems and that in most cases BBR does not share bandwidth in a fair manner with any of the tested algorithms like Reno, CUBIC, Vegas and Illinois.

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.

As BBR's underlying model is flawed in several aspects, drastic changes are proposed for BBR 2.0. Our framework aids the active development and improvement, as all experiments can easily be repeated and reproduced. This allows to easily verify the impact of changes to the algorithm, quantify improvements and avoid regressions. Our framework as well as the raw data for all figures presented is available online [3] 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).

The paper at hand is an extended version of our paper presented at IFIP 2018 conference [48]. This paper focuses more on the framework for reproducible measurements and expands the former explanations about congestion control and the framework. Furthermore, the evaluation of BBR's inter-protocol fairness has been extended significantly.

[1] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, Host-to-host congestion control for TCP, IEEE Communications surveys & tutorials 12 (3) (2010) 304–342.

[2] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, Y. Lu, TCP congestion avoidance algorithm identification, IEEE/Acm Transactions On Networking 22 (4) (2014) 1311–1324.

[3] Framework and Data Publication, URL `https://gitlab.lrz.de/tcp-bbr`, 2018.

[4] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, BBR: Congestion-based Congestion Control, ACM Queue 14 (5).

[5] S. Ha, I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, ACM SIGOPS Operating Systems Review 42 (5).

[6] 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.

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

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

[9] S. Ma, J. Jiang, W. Wang, B. Li, Towards RTT Fairness of Congestion-Based Congestion Control, CoRR abs/1706.09115, URL `http://arxiv.org/abs/1706.09115`.

[10] J. Gettys, K. Nichols, Bufferbloat: Dark Buffers in the Internet, Commun. ACM 55 (1), ISSN 0001-0782, doi:\bibinfo{doi}{10.1145/2063176.2063196}.

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

[12] L. S. Brakmo, L. L. Peterson, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal on selected Areas in communications 13 (8).

[13] 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.

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

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

[16] S. Liu, T. Başar, R. Srikant, TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks, Performance Evaluation 65 (6-7) (2008) 417–440.

[17] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data Center TCP (DCTCP), in: Proceedings of the 2011 ACM SIGCOMM Conference, vol. 41, ACM, doi:\bibinfo{doi}{10.1145/1851275.1851192}, 2011.

[18] K. Winstein, H. Balakrishnan, TCP Ex Machina: Computer-generated Congestion Control, in: Proceedings of the 2013 conference on ACM SIGCOMM Conference, vol. 43, ACM, doi:\bibinfo{doi}{10.1145/2534169.2486020}, 2013.

[19] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, M. Schapira, PCC: Re-architecting congestion control for consistent high performance, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015.

[20] L. Xu, K. Harfoush, I. Rhee, Binary increase congestion control (BIC) for fast long-distance networks, in: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, IEEE, 2004.

[21] J. Mo, R. J. La, V. Anantharam, J. Walrand, Analysis and comparison of TCP Reno and Vegas, in: INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 3, IEEE, 1556–1563, 1999.

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

[23] B. Lantz, B. Heller, 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.

[24] S. Floyd, Metrics for the Evaluation of Congestion Control Mechanisms, RFC 5166, RFC Editor, 2008.

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

[26] V. Jacobson, R. Braden, D. Borman, RFC 1323, TCP extensions for high performance .

[27] S. Ebrahimi-Taghizadeh, A. Helmy, 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.

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

[29] W. K. Leong, Z. Wang, 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] E. Atxutegi, F. Liberal, H. K. Haile, K.-J. Grinnemo, A. Brunstrom, A. Arvidsson, On the Use of TCP BBR in Cellular Networks, IEEE Communications Magazine 56 (3) (2018) 172–179.

[31] F. Li, J. W. Chung, X. Jiang, M. Claypool, TCP CUBIC versus BBR on the Highway, in: International Conference on Passive and Active Network Measurement, Springer, 269–280, 2018.

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

[33] D. G. Feitelson, From Repeatability to Reproducibility and Corroboration, SIGOPS Oper. Syst. Rev. 49 (1), ISSN 0163-5980, doi:\bibinfo{doi}{10.1145/2723872.2723875}.

[34] O. Bonaventure, April 2016: Editor's Message, in: ACM SIGCOMM CCR, 2016.

[35] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, G. Carle, High-Performance Packet Processing and Measurements (Invited Paper), in: 10th International Conference on Communication Systems & Networks (COMSNETS 2018), Bangalore, India, 2018.

[36] A. Brooks, J. Daly, J. Miller, M. Roper, M. Wood, Replication's role in experimental computer science, EfoCS–5–94 (RR/94/171) .

[37] V. Bajpai, M. Kühlewind, J. Ott, J. Schönwälder, A. Sperotto, B. Trammell, Challenges with Reproducibility, in: ACM SIGCOMM'17 Workshop on Reproducibility, 2017.

[38] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, G. Carle, Towards an Ecosystem for Reproducible Research in Computer Networking, in: ACM SIGCOMM'17 Workshop on Reproducibility, 2017.

[39] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible Network Experiments Using Container-based Emulation, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, ACM, ISBN 978-1-4503-1775-7, doi: \bibinfo{doi}{10.1145/2413176.2413206}, 2012.

[40] L. Yan, N. McKeown, Learning Networking by Reproducing Research Results, SIGCOMM Comput. Commun. Rev. 47 (2), ISSN 0146-4833, doi:\bibinfo{doi}{10.1145/3089262.3089266}.

[41] C. Paasch, R. Khalili, 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, CoNEXT '13, ACM, ISBN 978-1-4503-2101-3, doi:\bibinfo{doi}{10.1145/2535372.2535403}, 2013.

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

[43] C. A. Grazia, N. Patriciello, M. Klapez, 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.

[44] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, K. Winstein, Pantheon: the training ground for Internet congestion-control research, in: 2018 USENIX Annual Technical Conference (USENIX ATC 18), 731–743, URL `http://pantheon.stanford.edu`, 2018.

[45] J. Crichigno, Z. Csibi, E. Bou-Harb, N. Ghani, Impact of Segment Size and Parallel Streams on TCP BBR, in: 2018 41st International Conference on Telecommunications and Signal Processing (TSP), IEEE, 1–5, 2018.

[46] N. Cardwell, Y. Cheng, et al., BBR Congestion Control Work at Google IETF 102 Update, IETF 102 URL `https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-an-update-on-bbr-work-at-google-00`, Presentation Slides.

[47] N. Cardwell, Y. Cheng, et al., BBR Congestion Control Work at Google IETF 101 Update, IETF 101 URL `https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at-google-00`, Presentation Slides.

[48] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, G. Carle, Towards a Deeper Understanding of TCP BBR Congestion Control, in: IFIP Networking 2018, Zurich, Switzerland, 2018.