IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 27, 2020, accepted February 28, 2020, date of publication March 2, 2020, date of current version March 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977834

BBRp: Improving TCP BBR Performance Over WLAN

CARLO AUGUSTO GRAZIA, MARTIN KLAPEZ, AND MAURIZIO CASONI, (Senior Member, IEEE)
Department of Engineering Enzo Ferrari, University of Modena and Reggio Emilia, 41125 Modena, Italy
Corresponding author: Carlo Augusto Grazia (carloaugusto.grazia@unimore.it)

ABSTRACT This paper shows the inefficiency of TCP BBR in exploiting the Wi-Fi bandwidth. This limitation of BBR has been observed with both IEEE 802.11n and IEEE 802.11ac, where the mechanism of frame aggregation is used to boost the throughput of data transmission. In the last years, many TCP variants have been introduced to limit the bufferbloat phenomena and bound the latency through a reduction of the queue backlog injection rate. However, this mechanism impacts on the Wi-Fi frame aggregation logic, impeding TCP congestion controls to reach the full throughput potential of a Wi-Fi interface. While this problem can be solved with TCP Cubic by allowing the sender node to enqueue more packets, for TCP BBR the fix is not the same, as it has a customized pacing algorithm. With this contribution we propose BBRp, a new BBR version that allows for fine-tuning the congestion control pace, achieving between four and six times more throughput over IEEE 802.11n and IEEE 802.11ac channels, at the cost of an increased latency that is however always less than the latency obtainable with loss-based TCP congestion controls.

INDEX TERMS BBR, latency, TCP, TSQ, WLAN.

I. INTRODUCTION

The increase of Wi-Fi users is demanding new optimized standards, as well as refinements in the current ones [1]–[3]. Considering the two most used technologies for WLAN environments, namely IEEE 802.11n and IEEE 802.11ac, the concept of frame aggregation has been introduced to increase the overall throughput [4]. Simultaneously, several solutions to limit the end-to-end latency have been introduced to overcome the bufferbloat phenomena [5]. The developers of the Linux kernel have been very active in the last years introducing several features and modules related to TCP communications, with new congestion controls like BBR [6], proposed by Google LLC (Google) at the beginning of 2017. BBR is currently available on many Linux's distributions, Google has incrementally applied it to its YouTube servers, and it is in the process of being improved to BBR v2.0 in order to deal with several limitations highlighted by the research community in the last years [7]–[10]. Moreover, Google has also introduced a new mechanism called TCP Small Queues (TSQ) that limits the number of packets that a TCP socket can push down in the stack until packets have been truly dispatched by the Network Interface Card (NIC). Even if the TSQ performance over wired links is remarkable, this is not the case for WLAN environments in which TSQ could break the frame aggregation logic, impeding all the TCP variants to discover the full link potential correctly. This limitation has been discovered and solved for TCP Cubic, and it led to a new solution for boosting BBR v2.0 throughput on Wi-Fi paths [11]–[13]. Unfortunately, applying the same fix to the BBR algorithm is not enough to get a decent throughput from the Wi-Fi interface. The reason is that another essential TCP part of BBR is breaking the Wi-Fi frame-aggregation logic, i.e., TCP Pacing, which is a delay between the transmission of TCP segments. While all the TCP variants use a shared pacing structure, BBR has a customized one, hardcoded in the Linux kernel, that takes precedence over the default one.

A. CONTRIBUTION

In this paper, we highlight the challenges that arise combining TCP BBR, TSQ and TCP Pacing on a wireless bottleneck, providing real tests on several wireless technologies. We present a modified version of BBR called BBRp, able to exploit the Wi-Fi bottleneck without braking the frame-aggregation logic at the bottleneck. BBRp has been tested on a real test-bed, with a Wi-Fi access technology, over different use-cases and compared to several TCP congestion controls.

The associate editor coordinating the review of this manuscript and approving it for publication was Mubashir Husain Rehmani.

TABLE 1. Related work contributions summarized.

Related Work	Fairness	Wi-Fi	Ref.
Modest-BBR	Improve fairness with Cubic and loss-based congestion controls. Maintain a similar throughput performance with respect to BBR.	none	[10]
BBR-E	Improve short-RTT vs. long-RTT fairness in a simulation environment. It lacks in steadiness due to throughput oscillations.	none	[14]
BBQ	Reduce the end to end latency and boost (6x) the fairness with different RTT.	none	[15]
DA-BBR	It extends BBR-E by including the BBQ algorithm. It adds a dynamic RTT adaptation and extensive real-tests validation.	none	[16]
BBR-DEV	It is a less aggressive variant at the bottleneck with a smart and adaptive drain function.	It works only if the server interface is wired and the client one is a Wi-Fi (downlink).	[17]
TSQ Patch	none	Tailored for the uplink. It works only for loss-based variants.	[11]
BBRp	Fairness validated through the RRUL test in high-congested environments with uplink and downlink streams.	It solves the BBR Wi-Fi inefficiency for both the downlink and uplink regardless of the Wi-Fi bottleneck position.	[18]

We demonstrated the ability of BBRp to discover the full Wi-Fi bandwidth reaching almost optimal throughput values, outperforming the standard BBR algorithm, while still maintaining better latency performance when compared to TCP Cubic. A key contribution of BBRp is, indeed, to maximize the efficiency of a data transfer, increasing the throughput with a minimum latency increment, regardless of the bottleneck Wi-Fi position (i.e., proximal to the sender or the receiver). Moreover, the outcome of this research paved the way for BBR v2.0 and allowed users using long-term-support Linux kernels such as 4.14 and 4.19 to boost the BBR performance on the Wi-Fi path.

The rest of the paper is organized as follows: Section II describes the related work, while Section III enhances the critical points of a wireless bottleneck. Section IV presents the current TCP stack available on Linux systems and Section V describes the testbed used to produce the results available in Section VI. Finally, Section VII concludes the paper.

II. RELATED WORK

Several scientific contributions have analyzed BBR performance in different scenarios and technologies. As an example, some recent works have tried to answer this question: ''will TCP work in mmWave 5G cellular networks?''. One work is based on simulations and also includes BBR among the TCP congestion controls investigated [19]. A general issue for mmWave is that, for the currently available TCP variants, it is challenging to exploit the available bandwidth during ''irregular'' time intervals. Continuing on the cellular network topic, with currently available technology such as 4G, BBR has been tested in [20], [21]; both the papers conclude that TCP BBR outperforms TCP Cubic and TCP NewReno in terms of throughput and latency trade-off, but in some network conditions BBR struggles in maintaining fairness between flows. Another work confirms a latency reduction on mobile edge computing for BBR with respect to Cubic [22].

Concerning fairness, when different RTT flows are in place, BBR has been discovered to provide better treatment for higher-RTT flows, while another work shows that it is hard to achieve fairness between BBR and Cubic [14], [23], [24]; thanks to FQ-Codel, the unfairness gap can be reduced remarkably [25]. General performance on the impact of TCP BBR versus TCP Cubic traffic has also been investigated in [26]. A variant of BBR, called Modest-BBR, modifies BBR, reducing its aggressiveness and increasing fairness with Cubic while still maintaining similar performance to the original BBR [10]. Another variant of BBR called, instead, DA-BBR, focuses on the RTT-fairness achieving fair throughput between short-RTT flows and long-RTT flows where RTT is five times higher [16]. DA-BBR works on top of the BBQ algorithm, which has been the first attempt to address the problem of RTT fairness of the original BBR [15]. BBQ continuously detected the excess queues and limited the time for probing when a queue was created to prevent long RTT flows from transmitting a considerable amount of traffic to the pipe. To conclude the picture, two works investigate the behavior of mixed BBR and Cubic traffic dealing with the internal parameters of BBR, in particular with its cycle [27], [28].

Considering real tests of BBR over Linux systems, BBR and Cubic have been tested over standard Gigabit Ethernet wired networks with a 4.9 kernel version [8]. The paper shows that BBR does not meet its standard behavior when multiple flows are in place, both in terms of fairness and latency reduction due to high queue occupancy. On the other side, the frame aggregation over WLAN technologies has been investigated mainly through analytical models and simulations, initially on IEEE 802.11n and, recently, on IEEE 802.11ac [29], [30].

To the best of our knowledge, the only scientific contribution investigating the performance of BBR over WLANs has been proposed by the author of BBR, Neal Cardwell with a RFC, proposing a BBR patch that we name BBR-DEV hereafter [17]. BBR-DEV can operate, increasing the BBR throughput, in the case where the TCP sender is on Ethernet and the receiver is on a Wi-Fi network. To do it, BBR-DEV instructs the sender to put extra data in flight to keep the bottleneck utilized. Moreover, BBR-DEV introduces also an adaptive drain technique that has the goal of lowering queuing delays. We overcome the limitations of BBR-DEV, concerning the throughput increment on the Wi-Fi path, by proposing a solution that works also in the scenarios in which the sender is directly attached to a Wi-Fi interface. Our investigation

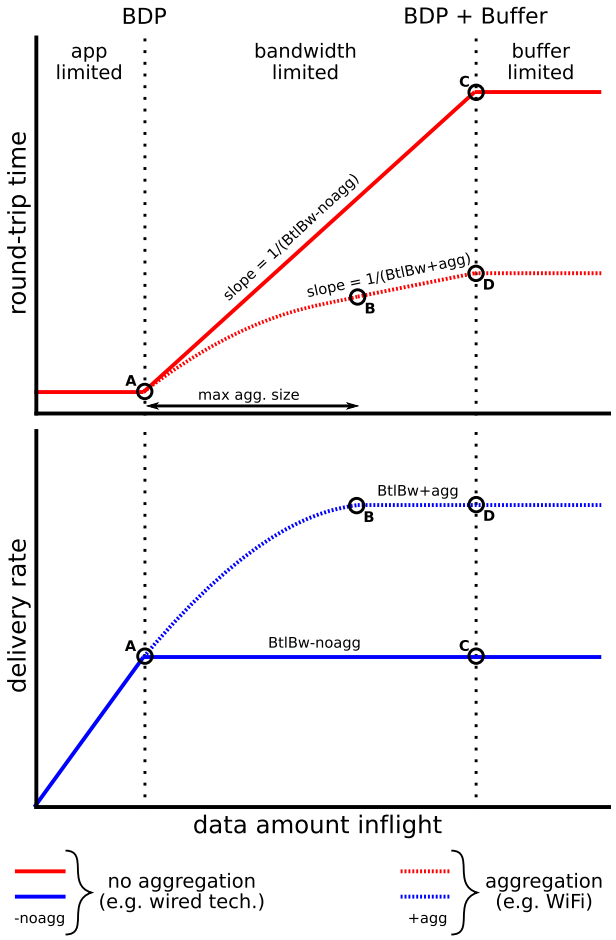


FIGURE 1. Data-rate and round-trip time as a function of the amount of data inflight: aggregation vs. non aggregation.

involves not only the congestion control alone but also other new mechanisms such as TSQ and TCP Pacing that both play a crucial role in the frame aggregation stage.

III. WIRELESS BOTTLENECK

To understand the limits of BBR, we refer to Figure 1 in which is depicted the difference between a standard wired bottleneck and a wireless 802.11n/ac one. The Figure reports the delivery rate and the Round-Trip Time (RTT) of one flow, according to the amount of data inflight. The typical RTT/throughput tradeoff, reported in bold, guided the design principle of all the "bufferbloat-oriented" solutions and have been presented in the original BBR paper [6]. Indeed, in wired networks, after reaching the point A (corresponding to the Bandwidth-Delay Product - BDP - known as Kleinrock's optimal operating point) of the Figure, increasing the amount of data inflight has the sole effect of increasing the RTT, due to the formation of queues of packets and queueing delays at the bottleneck buffer, while the delivery rate remains constant and equal to the bottleneck bandwidth (BtlBw). Another essential operating point for wired networks is C, which is the loss-based congestion control operating point, where the queue at the bottleneck gets full, and packets start to be dropped, allowing the sender to moderate the amount of data inflight once the losses are registered.

The assumptions behind this tradeoff, unfortunately, fall when considering a wireless bottleneck where the aggregation technique is enabled (this is the case of the standards IEEE 802.11n and 802.11ac). In a Wi-Fi bottleneck, having more than one packet enqueued is exploited opportunistically by aggregating data in a single large frame to reduce the protocol overhead and increase the efficiency of the transmission, increasing the ratio between the payload size and the total frame size. In other words, this is done by reducing the fixed MAC layer overhead and medium contention overhead, which results in less airtime consumption and higher throughput. This mechanism has an upper limit, known as the maximum aggregation size; after that, it is necessary to form a new aggregate, i.e., a new frame. Considering the aggregation mechanism, it is not valid anymore that enqueueing more packets has no impact on the delivery rate; indeed, the delivery rate increases as a function of the aggregation size, up to the maximum bottleneck bandwidth reached with the maximum aggregation size (BtlWw+agg) [31], [32]. Simultaneously, the RTT increases less than linearly, unlike that in wired bottlenecks, because the larger is the number of packets enqueued (i.e., the larger the aggregate), the higher will be the data rate, while the RTT increment will be lower, reducing the slope. This introduces a new optimal point B, which is the equivalent of A but for a wireless bottleneck. After reaching B, increasing the amount of data inflight has, again, the sole effect of increasing the RTT without increasing the throughput. Similarly, loss-based variants operate at the point D, which is the equivalent of C moving from wired to wireless bottlenecks, where losses are generated. An experimental validation of Wi-Fi bottleneck curves is provided in Section VI-A.

The most critical difference between the two bottlenecks, and the two optimal points A and B, is that in B, reaching the maximum delivery rate, comes at the expense of an RTT increment equal to the transmission time of a packet with the maximum aggregation size.

IV. STACK

This section describes the current TCP/IP stack of the Linux kernel, including all the new parts subject of this paper, like TSQ and TCP Pacing. To accurately present these new modules, the queueing discipline (QDisc) layer and the driver are also reported. Indeed, Figure 2 models the TCP Linux subsystem with the new features, the QDisc block, and the Driver block. The section also reports details on BBR and BBRp.

The current Linux TCP module is composed of three main algorithms, namely TCP Congestion Control, TCP Small Queues, and TCP Pacing. On top of this module, there is the TCP Socket, which manages the ACKs and deals with physical packets. Every TCP connection is mapped with a specific TCP socket, and the packets are managed according to the three algorithms.

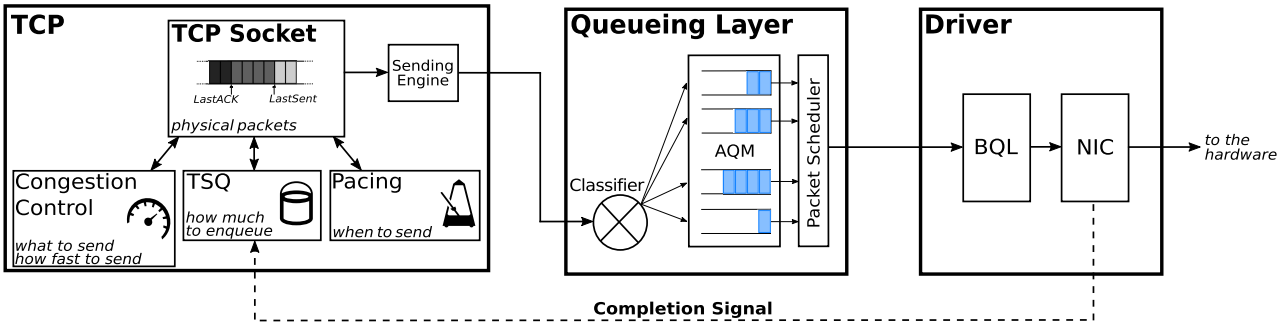


FIGURE 2. Linux TCP sender architecture.

Congestion Control: it is a well-known part of the TCP module, rich in literature contribution with many possible algorithms that can be used. In this paper are tested TCP Cubic, the current default Linux solution, TCP BBR, the congestion control algorithm designed by Google and incrementally deployed in many nodes, but also TCP New Reno and TCP New Vegas, a standard loss-based solution and a delay-based one respectively. These four algorithms are well different in terms of the approach: on one side we have BBR, a rate-based variant where the concept of time is stressed to reduce latency as the main goal, and New Vegas, a pure delay-based variant that presaged many elements of BBR; on the other side, instead, we have loss-based variants such as Cubic and New Reno. Each congestion control is responsible for fundamental operations like the computation of the sending rate and the congestion window size, as well as the computation of the TCP parameters in the presence of congestion events or packet losses.

TCP Small Queues (TSQ): it is the algorithm introduced by Google to mitigate each TCP flow latency. To achieve this result, each TCP socket is allowed to enqueue in the node stack a limited number of packets mitigating the Bufferbloat [5] phenomena and avoiding the accumulation of packets in the sender node queues; only when the NIC finalizes the dispatch of a packet, the TCP socket is informed and is allowed to enqueue a new packet in the stack. The standard TSQ behavior on wired networks is to allow each TCP socket to enqueue a number of packets that is equivalent to the number of packets that would be sent in 1 ms at the current sending rate; this mechanism helps in maintaining an upper bound of the queueing delay of the sending node as a function of the flow throughput. This global constraint of 1 ms has been proved in [11] to be too strict in a Wi-Fi environment where the frame aggregation is not possible with such a limit.

TCP Pacing: it is the algorithm that defines the pace used to push the packets from the TCP module to the lower layers of the stack. While the TSQ limits the number of packets enqueued, the TCP Pacing limits the internal rate for moving packets, forcing a time interval between an enqueue and another; in this sense, both TSQ and TCP Pacing help avoiding the formation of bursts mitigating the Bufferbloat effect.

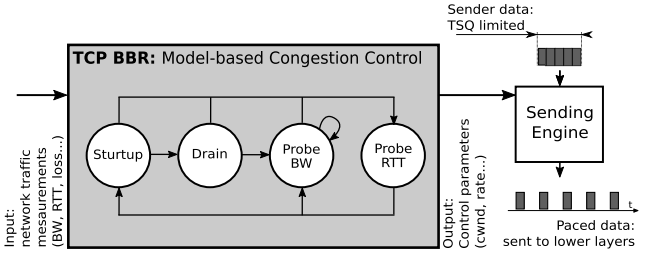


FIGURE 3. Linux TCP BBR block.

A standard TCP Pacing algorithm is used by almost all the TCP congestion controls except TCP BBR, which implements its own solution. The Linux kernel uses two default rates for pacing, expressed as a percentage of the current rate of a TCP flow. These two values are equal to 200% during the slow-start phase (allowing to enqueue packets at a rate which is twice the current one) and equal to 120% in the congestion avoidance phase (allowing to enqueue packets at a rate which is 20% higher than the current one). TCP BBR uses a similar value called TCP Pacing Gain, hardcoded in the BBR algorithm and not tunable in user space, equal to a rate which is 25% higher of the current one.

TCP BBR: if BBR is selected as TCP congestion control, the TCP block of Figure 2 behaves following the Figure 3 description. The core feature of BBR is to be model-based, and it behaves following a state-machine composed by four states: Sturtup, Drain, ProbeBW and ProbeRTT. The first two states belong to the initial part of a connection, then BBR moves to a steady-state phase composed by the last two states [16]. The input for the state machine are the last 10 RTT samples and the last estimated bandwidth, while the output consists of the congestion window size, and the pacing rate for the sending engine. The sending engine is the last step before enqueueing the packets in the lower layer of the stack, which is the QDisc one of Figure 2. The sending engine is similar to the mechanism used by all the TCP congestion controls, the difference is that all the algorithms, except BBR, use global pacing rate, while BBR uses the one provided by its model, and it cannot be tuned in user-space. During the sending engine stage, each TCP congestion control knows

the amount of data that can be enqueued, limited by the congestion window size and, in particular, by the TSQ policy, and then the data is delivered to the lower layers pacing it at the current pacing rate. The higher the pacing rate, the shorter will be the distance, in terms of time, between the packets in the final stage of Figure 3.

TCP BBRp: It differs from BBR for the ProbeBW state. The ProbeBW is the most critical one in terms of bandwidth estimation, indeed BBR spends most of the time in this state to probe the available bandwidth. The ProbeBW state consists of eight cycles in which the `pacing_gain` variable assumes the eight values, cyclically, of [1.25, 0.75, 1, 1, 1, 1, 1, 1]. At the first cycle, the standard BBR `pacing_gain` is placed at 1.25, which is not enough to guarantee a proper bandwidth estimation if the bottleneck uses aggregation policies. The general idea of the first cycle is to send more data to check if the available bandwidth is greater than the one estimated so far. Continuing, the subsequent value of the `pacing_gain` variable is 0.75, to drain the bottleneck queue by removing the excess of packets generated by the previous probing cycle. The remaining six cycles consist, instead, of a steady-state in which data is sent at a constant `pacing_gain` of 1 with the bottleneck bandwidth obtained in the previous probing cycle. The critical change in the ProbeBW state of BBRp is described in Algorithm 1, where the standard first value of 1.25 for the `pacing_gain` is replaced by the `bbrp_pace` variable. Indeed, our patch allows to tune the first `pacing_gain` value for tuning bandwidth probe. In our test we used 1.5 instead of 1.25, providing a `bbrp_pace` of 6. We remind that BBRp falls-back to the standard BBR behavior with a `bbrp_pace` of 5. The BBRp patch is available, together with our open-data, with more tests with different `bbrp_pace` values [18].

Algorithm 1 BBRp Algorithm.

```
Input: UNIT, bbrp_pace
1: int pacing_gain[] = {
       UNIT × bbrp_pace / 4,          // probe for more bw
       UNIT × 3 / 4,                  // drain queue
       UNIT, UNIT, UNIT,              // cruise at bw
       UNIT, UNIT, UNIT               // without bloating
   };
   [...]
2: bw = get_bbr_max_bw();
3: min_rtt = get_bbr_min_rtt();  // BBR model parameters
4: if pacing_gain > UNIT then
5:     cwnd = bw × min_rtt × pacing_gain;  // BDP × gain
6: end if
```

Once the TCP socket generates a packet, the packet is moved into the QDisc layer of Figure 2 that can be both a separate block in case of a wired connection, as well as a black-box integrated into the driver as is the case with the `ath9k` and `ath10k` drivers. Figure 2 reports the standard structure of the FQ-Codel [25] algorithm as the default option of many Linux distributions. The last block is the driver, the piece of code that interacts with the Network Interface Card (NIC) and deliver packets on the medium. A very last queue is present in the driver; it is typically a FIFO and is ruled by a Byte Queue Limit (BQL) [33], [34] to avoid excessive queueing. This limit is hard-coded in the kernel and not part of our tests.

FIGURE 4. Physical testbed layout.

TABLE 2. Testbed parameters.

parameter	value
Kernel version	4.19-lts
Linux Distribution	Arch Linux
TCP Congestion Control	BBR, BBRp, New Vegas Cubic, New Reno
TSQ type	`TSQ` (standard), `2TSQ` `4TSQ` `8TSQ`
Queueing discipline	FQ_Codel
Wireless Chipset	Atheros AR9271 1x1 MIMO (USB Dongle) Atheros AR5BHB116 2x2 MIMO Qualcomm QCA6178 1x1 MIMO Qualcomm QCA9880v2 2x2 MIMO
Wireless Driver	`ath9k-htc` for IEEE 802.11n (USB Dongle) `ath9k` for IEEE 802.11n `ath10k` for IEEE 802.11ac
Tests	1-8 TCP Uploads\Downloads, RRUL
Metrics	TCP Throughput ICMP Latency (`ping` RTT)

V. TESTBED

This section describes our testbed, which is depicted in Figure 4. Each test involves three nodes, one wireless client, one wired server, and the router in the middle that provides the connectivity to both the other nodes. All the nodes deploy Arch Linux as the operating system with a 4.19 kernel version. This testbed represents a typical home/office connection with a desktop or a laptop connected to a Wi-Fi Access Point using the IEEE 802.11n or IEEE 802.11ac standard, while the rest of the network is then typically wired as in our case. The wireless connectivity is given by PCIe Atheros chipsets supported by the `ath9k` and `ath10k` open drivers.

This testbed allows configuring typical connections with different bottleneck positions. One example is a fast home/office connection with 1 Gbit/s backhaul that suffers a local bottleneck, which is the wireless interface between the client and the access point. Another example, instead, considers a backhaul of 100 Mbit/s imposed by the Internet Service Provider (ISP), and the wireless access network is not the bottleneck anymore, which is, instead, represented by the wired connection between the access point and the server.

The client uses one of the five possible TCP congestion control algorithms reported in Table 2, namely: BBR, BBRp, New Vegas, Cubic, and New Reno. The client can also set

different possible TSQ limits [11], [13]: it can be the standard dynamic value of 1 ms of data at the current rate, or it can be relaxed with the configuration of `2TSQ`, `4TSQ` and `8TSQ` that accommodate 2 ms, 4 ms and 8 ms of data, respectively. The incremental steps follow the powers of 2 because, at the kernel level, the TSQ size is managed as a bits shift operation.

The most critical parameter introduced in this paper is the TCP Pacing rate. Since BBR does not react to any modification to the standard pacing value offered by the current Linux systems, we patched it, exposing the TCP Pacing Gain variable used internally. We named this patched version BBRp and changed the default pacing rate, increasing its value, moving from the standard BBR TCP Pacing Gain equal to a rate which is 25% higher of the current one, to a BBRp TCP Pacing Gain equal to 50%. BBRp enables a fine-grained tuning of the pacing rate and details about our BBRp patch, the results not included here for space limitations, and several other possible pacing ratios can be found in [18]. The proposed solution is steering the design of BBR v2.0 [7].

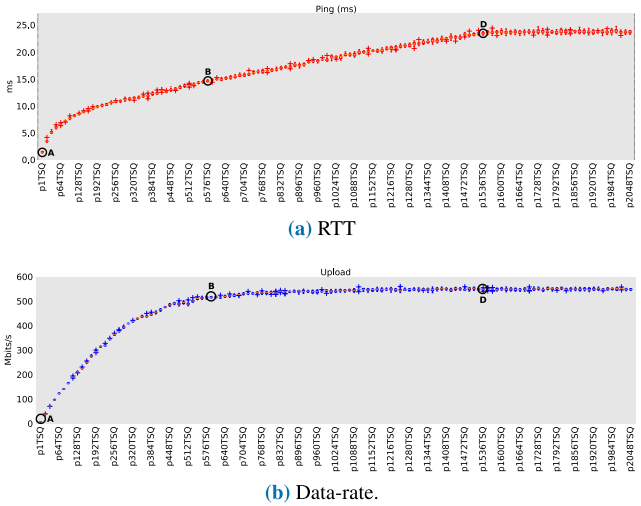
All the experiments reported in this paper have been organized by using the Flent [35] tool, a flexible network tester that gives the possibility to manage different traffic typologies efficiently as well as to collect many performance metrics. Tests are organized as follows. We start a standard TCP traffic in upload or download between the wireless client and the server. Each test runs for 40 seconds, 5 initial seconds with only ICMP traffic, 30 middle seconds in which also the actual TCP transmission is performed, and 5 final seconds where, again, only the ICMP traffic is maintained. In this way, it is possible to highlight the impact of the TCP traffic on the `ping` RTT, as well as many other parameters related to the TCP traffic itself, like throughput and TCP RTT. Similar to the simple TCP upload or download, the Realtime Response Under Load (RRUL) test is designed in the same way, but it uses 4 TCP flows in upload, and other 4 TCP flows in download, all simultaneously active during the 30 central seconds of each test. The parameters used to configure the testbed of our experiments are summarized in Table 2.

VI. RESULTS

This section reports the results collected during our experiments. A first suite of test validate the Wi-Fi bottleneck curve of Figure 1 through experimental analysis then, the rest of the result section is divided into three groups, one for each experiment investigated: TCP upload, TCP download, and RRUL, used for analyzing network performance under the heavy workloads that typically induce bufferbloat and other networking problems. Each experiment has been replicated 10 times. The recorded data are reported as candlesticks, which are the result of the aggregation of each iteration. The central box of each candlestick reports the 10th and 90th percentiles, while the horizontal line inside each box represents the median value.

A. WI-FI BOTTLENECK VALIDATION

To validate Figure 1, we provided experimental analysis by running a single TCP New Reno upload varying the TSQ size at the sender side. To do so, we used a feature of our TSQ patch which allows to control the amount of data at the sender side in a static way, controlling the amount of packets that the TCP socket can enqueue, regardless of the flow data-rate; this ease the comparison with the theoretical curve seen in Section I that is usually plotted as a function of the data inflight. Figure 5 shows the result of this experimental analysis dividing the output of the RTT, plotted in red in Figure 5a, to the output of the bandwidth, plotted in blue in Figure 5b. The experiment validates the bandwidth-limited and the buffer-limited area of Figure 1. Moreover, we enhanced on Figure 5 the operating points that correspond to `A`, `B` and `D` of Figure 1.



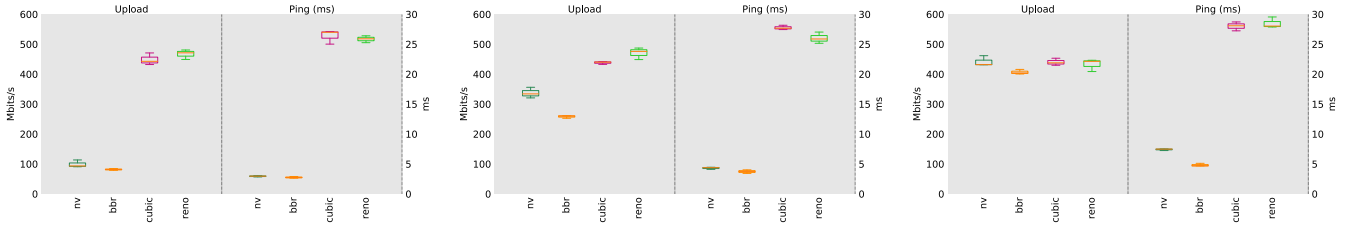
(a) RTT

(b) Data-rate.

FIGURE 5. Data-rate and round-trip time as a function of the data inflight: experimental data on IEEE 802.11ac.

B. TCP UPLOAD

One critical problem that we have observed and solved in [11] is related to the TCP upload in an IEEE 802.11n/ac network. Indeed, the recently adopted TSQ mechanism breaks the aggregation logic at the local Wi-Fi bottleneck, and this is the reason why we did different experiments considering different TSQ sizes, to relax the limit on the number of packets to be enqueued and, consequently, boost the throughput. We used a single TCP flow in upload because it is the most challenging scenario in which a single flow is in charge of exploiting the entire bandwidth of the Wi-Fi bottleneck. To demonstrate it, we present in Figure 6 the global throughput reached by one, four and eight simultaneously active TCP flows in Figures 6a, 6b and 6c, respectively. In this experiment we disabled the TSQ logic, and so there is no limitation in the amount of data that each socket can enqueue in the node. The higher is the amount of flows competing for the Wi-Fi bottleneck, the higher is the amount of packets that the NIC can use to exploit the channel bandwidth through forming large aggregate frames. This moves the focus to a simple worst-case scenario: a single TCP aiming to use the entire wireless bandwidth.



(a) One flow. (b) Four flows (c) Eight flows.

FIGURE 6. Standard TCP upload: one, four and eight flows.

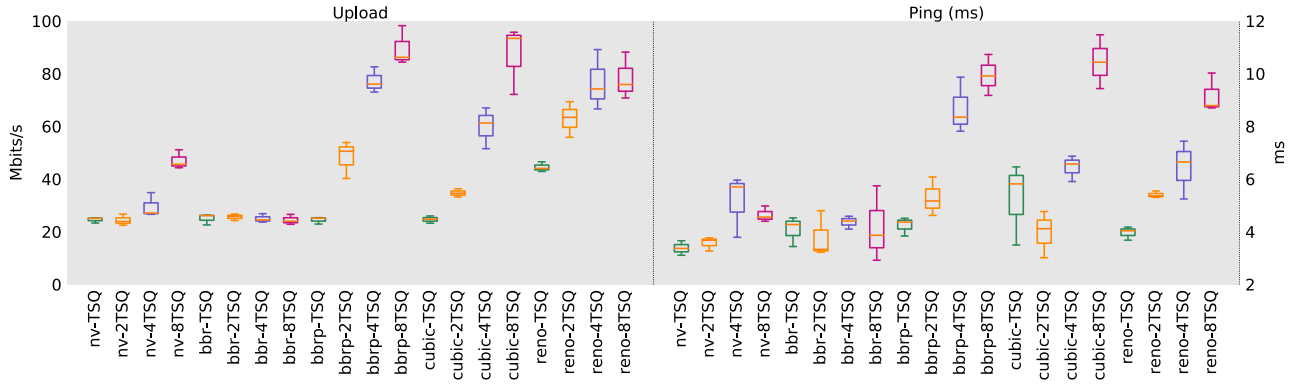


FIGURE 7. One TCP flow in upload with different TCP & TSQs: Goodput vs `ping`, `ath9k-htc`.

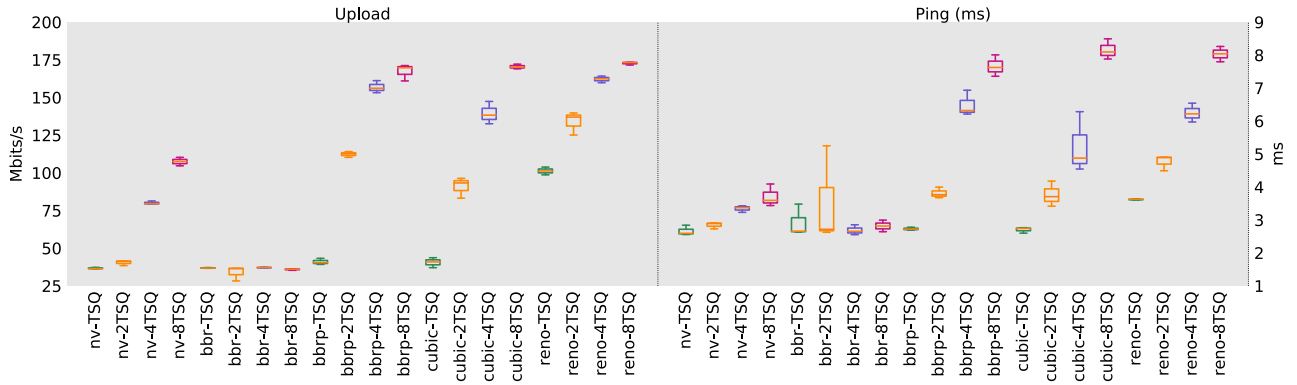


FIGURE 8. One TCP flow in upload with different TCP & TSQs: Goodput vs `ping`, `ath9k`.

Our solution previously proposed in [11] worked properly for loss-based congestion controls, but BBR did not react with a throughput increment when relaxing the TSQ limits, due to a mild TCP pacing rate, unable to support the aggregation logic at the bottleneck. This can be observed in the three Figures 7, 8, and 9. The difference between these Figures is the chipset used to create the Wi-Fi connectivity:

- In Figure 7 it is used the chipset Atheros AR9271 1 × 1 MIMO, which is a USB dongle, and due to this, the Linux kernel manages the wireless connection with the `ath9k-htc` driver, that deals with IEEE 802.11n connectivity.
- In Figure 8 it is used the chipset Atheros AR5BHB116 2 × 2 MIMO, which is a PCIe wireless card managed with the `ath9k` driver by the Linux kernel to deal, again, with IEEE 802.11n networks.
- In Figure 9, instead, it is mounted a Qualcomm QCA6178 1 × 1 MIMO, a PCIe wireless card able to create or join IEEE 802.11ac Wi-Fi networks through the `ath10k` driver.

These Figures, compared together, report that BBR is unable to boost the throughput even by relaxing the TSQ constraints. The reason is the pacing rate adopted by BBR that impedes the formation of bursts at the bottleneck, which in this case is clearly the wireless sender interface, with the consequence of breaking the Wi-Fi aggregation logic forcing the inability to increase the throughput. The other loss-based variants, Cubic and New Reno, quickly reach

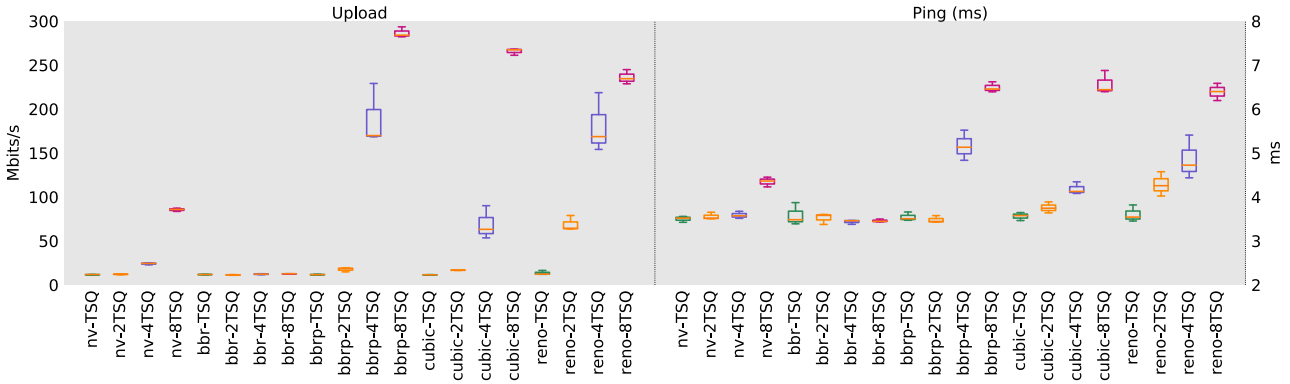


FIGURE 9. One TCP flow in upload with different TCP & TSQs: Goodput vs `ping`, `ath10k`.

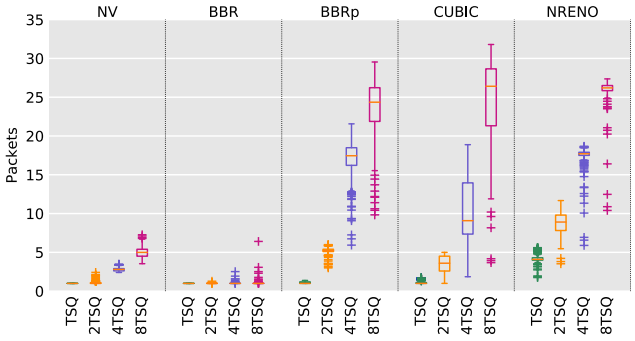


FIGURE 10. One TCP flow in upload with different TCP & TSQs: Wi-Fi aggregation size, `ath9k`.

their respective maximum throughput by relaxing the TSQ constraints. The price to pay for this throughput increment, as seen in Section III, is an RTT increment measured with the ICMP `ping` protocol. Even New Vegas, which is the delay-based congestion control at the base of the BBR model, is able to increase and even double its throughput with all the technologies. This because the pacing rate adopted by New Vegas is the same pacing rate adopted by all the congestion controls except BBR. As a matter of fact, New Vegas can not achieve the maximum throughput because it limits the RTT increments at 1 ms, limiting the data delivery-rate after this threshold. BBRp is reported in the center of each figure; thanks to our patch that increases the pacing rate, increasing its value, BBRp is able to increase the Wi-Fi upload throughput like loss-based congestion controls, up to the optimal values, without significative latency reductions nor increments. The distance between the minimum and the maximum RTT is limited at 5 ms by the FQ-CoDel queueing discipline used in the sender node. We chose to use a 1.5 factor for the pacing rate by analyzing the tradeoff of throughput and delay with different wired and wireless technologies. Indeed, our BBRp patch allows us to fine-grained tuning the pacing rate to the desired value, as can be observed in the data source [18].

To validate our analysis, we also report Figure 10, which includes the Wi-Fi aggregation size registered by the sender interface during the experiment of Figure 8. We included such data because, with the `ath9k` driver, the possibility to collect the aggregation statistics is enabled by default. With this Figure, it is possible to notice how loss-based congestion controls and BBRp can increase the aggregation size as a function of the TSQ limit; New Vegas increases the aggregation size as well but sharply limiting the maximum reachable aggregation, while, as a last conclusion, BBR never aggregates more than one packet, excluding few statistical outliers unable to steadily boost the throughput.

We reported first the TCP upload experiments because it is chronologically the first problem that we have dealt with as a consequent outcome of our previous work on TSQ [11]. Our patch has been included in the Linux kernel, and now `ath9k` and `ath10k` relax by default the TSQ limit at 4 ms of data.

We also considered a different scenario in which the wired connectivity between the access point and the server is limited at 100 Mbit/s by the ISP. This situation lets to migrate the bottleneck position from the wireless interface of the access network to the wired backhaul one. The same experiment described before has been run on this second scenario, and results are reported in Figure 11. One key characteristic of BBRp, in this scenario, is the ability to reach the 100 Mbit/s provided by the bottleneck with just a TSQ value equal to 2 ms, without an excessive queuing delay, while TCP Cubic needs a TSQ value equal to 4 ms to obtain the same result. Even TCP New Vegas is able to get close to the bottleneck bandwidth relaxing the TSQ constraints, and, as seen before, only BBR is not able to exploit the capacity of the path, saving 1 ms of `ping` RTT, but paying the price of a throughput well below 40 Mbit/s.

C. TCP DOWNLOAD

For what concerns a TCP download, we considered our Testbed in Figure 4 to model a standard scenario in which the server is connected through a Gigabit Ethernet to the Access Point, forming a reliable high-speed network, in which the sender (the server) does not have any TSQ issues related to aggregation. Indeed, in this case, we do not need to take the TSQ as a testing parameter because it does not affect the

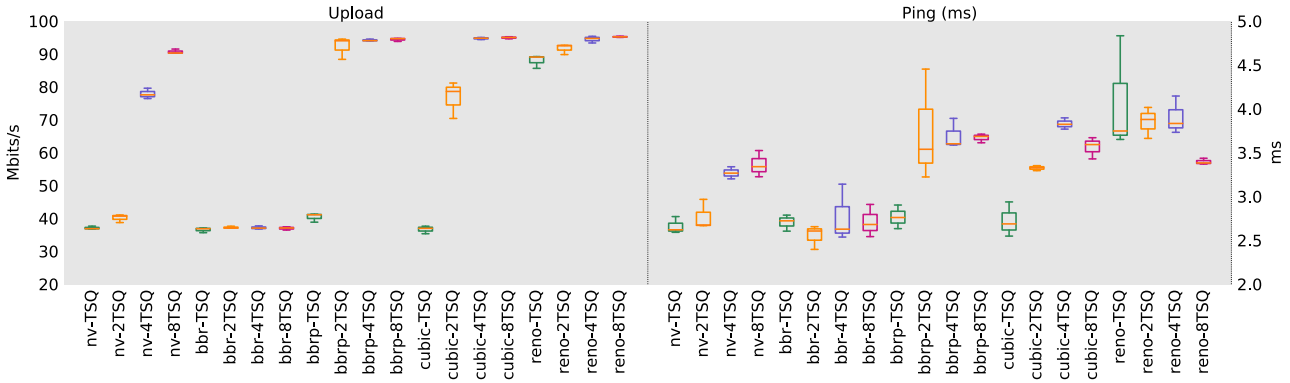


FIGURE 11. One TCP flow in upload with different TCP: Goodput vs `ping`, `ath9k` and 100 Mbit/s wired bottleneck.

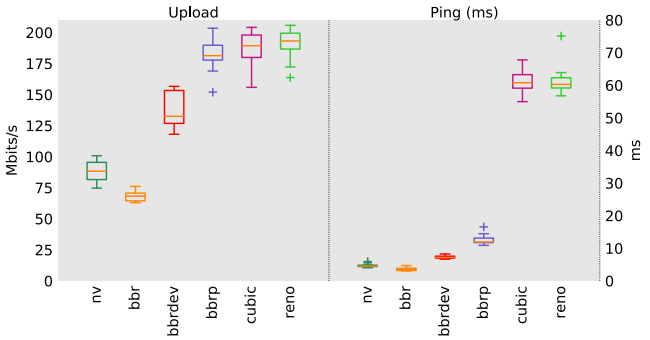


FIGURE 12. One TCP flow in download with different TCP: Goodput vs `ping`, `ath9k`.

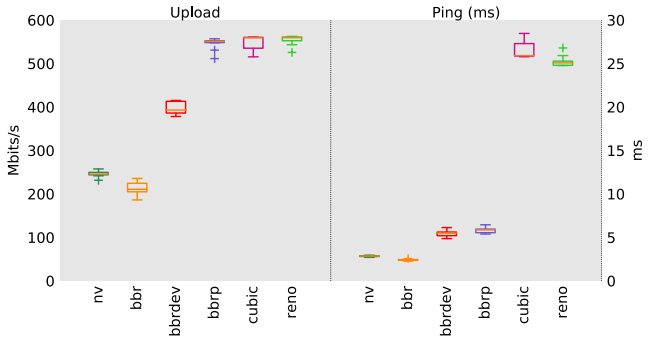


FIGURE 13. One TCP flow in download with different TCP: Goodput vs `ping`, `ath10k`.

communication in terms of TCP throughput. Instead, is the TCP Pacing difference between BBR and BBRp that affects the throughput by interacting with the aggregation logic of the remote Wi-Fi bottleneck. Indeed, from the point of view of the server, which is transmitting the TCP stream, the bottleneck is not the local interface, so it is clearly the typical condition in which the bottleneck is a remote segment of the end-to-end path, and there is no trivial control of it with back-pressure mechanisms. Moreover, this scenario allows a fair comparison with the BBR-DEV algorithm, because of the position of the wireless bottleneck not directly connected to the transmitting node [17].

The measurements collected during a single TCP download are reported in the two Figures 12 and 13. The difference between these Figures is, even in this case, the chipset used to create the Wi-Fi connectivity: Figure 12 uses the chipset Atheros AR5BHB116 2 × 2 MIMO, while Figure 13 uses a Qualcomm QCA9880v2 2 × 2 MIMO chipset. To enhance the difference between the TCP congestion controls performance, we configured the Access Point to use the `pfifo_fast` queueing discipline at the wireless bottleneck, which allows appreciating the different operating points of each TCP variant with respect to the wireless bottleneck model in Figure 1 of Section III.

The first important thing to notice is the inefficiency of TCP BBR and New Vegas to exploit the wireless

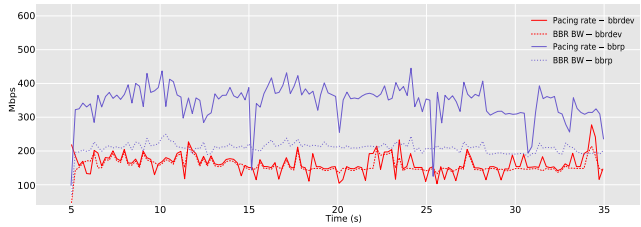


FIGURE 14. One TCP flow in download, BBR-DEV vs BBRp: pacing rate and BBR BW model.

bottleneck bandwidth. Indeed, TCP New Vegas reaches less than 100 Mbit/s in Figure 12 and 250 Mbit/s in Figure 13 with an 802.11n and 802.11ac wireless bottleneck, respectively. Similarly, BBR reaches even lower values of 75 Mbit/s in Figure 12 and 200 Mbit/s in Figure 13, respectively. These behaviors, recalling the wireless bottleneck discussion of Section III, correspond to the operating point A of Figure 1.

The second important thing to notice is that TCP loss-based variants, i.e., TCP Cubic and TCP New Reno, are able to boost the throughput close to the optimal limit imposed by the wireless bottleneck, which is slightly less than 200 Mbit/s in Figure 12 and 550 Mbit/s in Figure 13. The drawback is the RTT registered by these two congestion controls; indeed, the operating point of these loss-based variants is marked as D in Figure 1.

A third important thing is to acknowledge the improvement introduced by the BBR-DEV algorithm, the improved version of BBR that mitigates the performance limitation of the original BBR algorithm on Wi-Fi links. BBR-DEV almost doubles the data-rate of standard BBR in both the technologies investigated, introducing a queueing delay smaller than BBRp in the `ath9k` scenario and equal to BBRp in the `ath10k` scenario.

The fourth and final thing to notice is the almost optimal tradeoff of TCP BBRp in both the IEEE 802.11n and IEEE 802.11ac scenarios. In fact, in Figure 12, BBRp reaches 150 Mbit/s with less than 20 ms of RTT while, in Figure 13, it reaches 400 Mbit/s with less than 10 ms of RTT. Comparing BBRp to BBR, the former triplicates the throughput in both cases with smaller impacts on latency increments. This happens because BBRp works at the operating point `B` of Figure 1, increasing the throughput as soon as it is possible while impeding an RTT increment when the bottleneck bandwidth is reached.

Concluding the comparison between BBR-DEV and BBRp, Figure 14 reports the pacing rates (solid lines) and the BBR bandwidth values (dashed lines) collected during the IEEE 802.11n test. BBRp is able to maintain a higher bandwidth to fully exploit the Wi-Fi bottleneck capacity, with a data rate that is 100 Mbps higher than BBR-DEV; to do so, BBRp maintains a pacing rate between 300 and 400 Mbps in order to keep the bandwidth on 200 Mbps. Furthermore, a key characteristic of BBR-DEV is visible in the Figure: the absence of spikes corresponding to the draining phases. While BBRp manifests spikes of pacing rate reduction in conjunction with the 15th and 25th second of test (BBR base model, in fact, drains the queues every 10 seconds of activity), BBR-DEV does not reproduce the same trend, because of an adaptive drain mechanism.

D. RRUL TEST

To conclude our experimental section, we report here the results obtained by the different TCP congestion controls during an RRUL test. The testbed is configured exactly like the previous experiment, the TCP download, and we report in Figure 15 only the results obtained in the IEEE 802.11n scenario, for brevity. Considering that the test involves both 4 TCP streams in download and 4 TCP streams in upload, we configured the client to operate with the current Linux default TSQ value at 4 ms. In general, the unfairness between the download path and the upload path is clear, with the download streams that take a higher portion of the wireless bottleneck bandwidth. This is a consequence of the TSQ behavior, which is still a limit, even if it has been relaxed, of 4 ms for the upload path, while it is not a limit for the download path where the server, connected to the Access Point through a wired interface, can increase the throughput more easily. This characteristic has already been observed with TCP Cubic in [11], and Figure 15 confirms the trend also for the other congestion controls. The only exception is BBRp, which registers remarkable results. BBRp is the sole

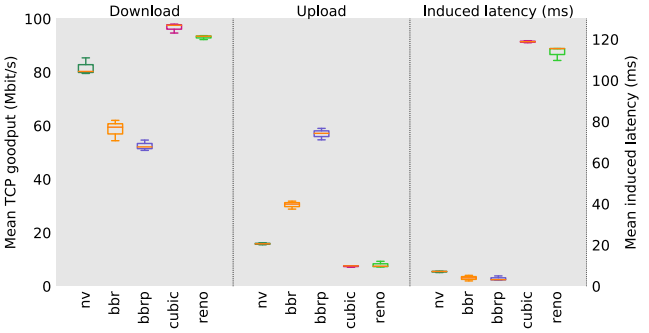


FIGURE 15. **RRUL test: Goodput vs** `ping`, `ath9k`, `4TSQ`.

TCP congestion control that guarantees fairness between the upstream and the downstream and, at the same time, it is able to limit the latency to the value of BBR and New Vegas at less than 10 ms. As a last note, even in this case, TCP loss-based variants suffer a high RTT due to their operating point `D` of Figure 1.

VII. CONCLUSION

This paper showed the inefficiency of TCP BBR over IEEE 802.11n and IEEE 802.11ac, the two most used Wi-Fi technologies. The reason for this inefficiency lies in the impossibility of performing frame aggregation with the standard BBR algorithm. We then introduced BBRp, which permits to tune the BBR pacing speed, allowing the congestion control to correctly aggregate packets at the wireless bottleneck and exploit the bottleneck bandwidth. Our experiments let us validate the BBRp performance in different scenarios concluding that our proposed variant solves the BBR inefficiency, reaching almost optimal TCP throughput while maintaining better performance in terms of latency when comparing it with both BBR-DEV and TCP loss-based variants like Cubic or New Reno. We demonstrate that BBRp behaves remarkably in several scenarios, considering TCP uploads with a local wireless bottleneck or a remote wired bottleneck, TCP downloads with a remote wireless bottleneck, and challenging RRUL scenarios with a highly congested environment. In particular, we have proved through real tests that BBRp increases the BBR throughput between 3 and 6 times over both the IEEE 802.11n and IEEE 802.11ac technologies, while preserving fairness by balancing the upstream and the downstream paths; simultaneously, BBRp minimizes the ICMP latency to values lower than those of TCP New Vegas and the standard TCP BBR.

REFERENCES

[1] E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi, "A tutorial on IEEE 802.11ax high efficiency WLANs," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 197–216, 1st Quart., 2019.

[2] B. Bellalta, "IEEE 802.11ax: High-efficiency WLANS," *IEEE Wireless Commun.*, vol. 23, no. 1, pp. 38–46, Feb. 2016.

[3] M. M. Islam, M. S. A. Mamun, N. Funabiki, and M. Kuribayashi, "Dynamic access-point configuration approach for elastic wireless local-area network system," in *Proc. 5th Int. Symp. Comput. Netw. (CANDAR)*, Nov. 2017, pp. 216–222.

[4] S. Das, P. Kar, and S. Barman, "Analysis of IEEE 802.11 WLAN frame aggregation under different network conditions," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Mar. 2017, pp. 1240–1245.
[5] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Queue*, vol. 9, no. 11, p. 40, Nov. 2011.
[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, pp. 58–66, Jan. 2017.
[7] N. Cardwell, "BBR V2: A model-based congestion control," in *Proc. ICCRG at IETF 104th Meeting*, Mar. 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00
[8] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.
[9] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, May 2018, pp. 109–117.
[10] Y. Zhang, L. Cui, and F. P. Tso, "Modest BBR: Enabling better fairness for BBR congestion control," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 646–651.
[11] C. A. Grazia, N. Patriciello, T. Hoiland-Jorgensen, M. Klapez, M. Casoni, and J. Mangues-Bafalluy, "Adapting TCP small queues for IEEE 802.11 networks," in *Proc. IEEE 29th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2018, pp. 1–6.
[12] C. Grazia and N. Patriciello. (Jun. 2018). *TCP Small Queues and WiFi Aggregation—A War Story*. [Online]. Available: https://lwn.net/Articles/757643/
[13] C. A. Grazia, "IEEE 802.11n/AC wireless network efficiency under different TCP congestion controls," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 1–6.
[14] G.-H. Kim, I. Mahmud, and Y.-Z. Cho, "Fairness improvement of BBR congestion control algorithm for different RTT flows," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Jan. 2019, pp. 1–2.
[15] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," 2017, *arXiv:1706.09115*. [Online]. Available: http://arxiv.org/abs/1706.09115
[16] G.-H. Kim and Y.-Z. Cho, "Delay-aware BBR congestion control algorithm for RTT fairness improvement," *IEEE Access*, vol. 8, pp. 4099–4109, 2020.
[17] N. Cardwell. (Apr. 2018). *Linux TCP BBR Patch for Higher WiFi Throughput and Lower Queuing Delays*. [Online]. Available: https://groups.google.com/forum/#!topic/bbr-dev/8pgyOyUavvY
[18] (Apr. 2019). *BBR+: Linux Kernel Patch, Source Scripts and Tests*. http://netlab.ing.unimo.it/sw/BBRp.zip
[19] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi, "Will TCP work in mmWave 5G cellular networks?" *IEEE Commun. Mag.*, vol. 57, no. 1, pp. 65–71, Jan. 2019.
[20] A. Parichehreh, S. Alfredsson, and A. Brunstrom, "Measurement analysis of TCP congestion control algorithms in LTE uplink," in *Proc. 2nd Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2018, pp. 1–8.
[21] E. Atxutegi, F. Liberal, H. K. Haile, K.-J. Grinnemo, A. Brunstrom, and A. Arvidsson, "On the use of TCP BBR in cellular networks," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 172–179, Mar. 2018.
[22] Z. Wang, Y. Tan, and X. Zhang, "Experimental evaluation of modern TCP variants in MEC-enabled cellular networks," in *Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2018, pp. 1–5.
[23] G.-H. Kim, Y.-J. Song, I. Mahmud, and Y.-Z. Cho, "Enhanced BBR congestion control algorithm for improving RTT fairness," in *Proc. 11th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2019, pp. 358–360.
[24] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda, and S. Yamaguchi, "TCP fairness among modern TCP congestion control algorithms including TCP BBR," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–4.
[25] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. (Jan. 2018). *FlowQueue-CoDel*. [Online]. Available: https://tools.ietf.org/html/rfc8290
[26] P. Hurtig, H. Haile, K.-J. Grinnemo, A. Brunstrom, E. Atxutegi, F. Liberal, and A. Arvidsson, "Impact of TCP BBR on CUBIC traffic: A mixed workload evaluation," in *Proc. 30th Int. Teletraffic Congr. (ITC)*, Sep. 2018, pp. 218–226.
[27] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cycle and divergence of performance on TCP BBR," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–6.
[28] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cyclic performance fluctuation of TCP BBR," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 811–812.
[29] Y. Lin and V. W. S. Wong, "WSN01-1: Frame aggregation and optimal frame size adaptation for IEEE 802.11 n WLANs," in *Proc. IEEE Globecom*, Nov. 2006, pp. 1–6.
[30] T. Moriyama, R. Yamamoto, S. Ohzahata, and T. Kato, "Frame aggregation size determination for IEEE 802.11ac WLAN considering channel utilization and transfer delay," in *Proc. 14th Int. Joint Conf. e-Bus. Telecommun.*, vol. 6, 2017, pp. 89–94.
[31] D. Skordoulis, Q. Ni, H.-H. Chen, A. P. Stephens, C. Liu, and A. Jamalipour, "IEEE 802.11n MAC frame aggregation mechanisms for next-generation high-throughput WLANs," *IEEE Wireless Commun.*, vol. 15, no. 1, pp. 40–47, Feb. 2008.
[32] T. Y. Arif and R. F. Sari, "Throughput estimates for A-MPDU and block ACK schemes using HT-PHY layer," *J. Comput.*, vol. 9, no. 3, pp. 678–687, Mar. 2014.
[33] J. Corbet. (Aug. 2011). *Network Transmit Queue Limits*. [Online]. Available: https://lwn.net/Articles/454390/
[34] N. Mareev, D. Kachan, K. Karpov, D. Syzov, and E. Siemens, "Efficiency of BQL congestion control under high bandwidth-delay product network conditions," in *Proc. Int. Conf. Appl. Innov. IT*, vol. 7, no. 1, pp. 19–22, 2019.
[35] T. Hoeiland-Joergensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," in *Proc. 11th EAI Int. Conf. Perform. Eval. Methodol. Tools*, 2017, pp. 120–125.

CARLO AUGUSTO GRAZIA received the Ph.D. degree from the Department of Engineering Enzo Ferrari (DIEF), University of Modena and Reggio Emilia (UNIMORE), in 2016. He is currently an Assistant Professor holding the course automotive connectivity with DIEF, UNIMORE. He has been involved in the EU FP7 Projects E-SPONDER and PPDR-TC. His research interests include computer networking, with an emphasis on wireless networks, queueing algorithms, and V2X.

MARTIN KLAPEZ received the Ph.D. degree from DIEF, UNIMORE, in 2017. He is currently a Postdoctoral Research Fellow with DIEF, UNIMORE. He has collaborated with the Italian Nanoscience National Research Center S3, where he has been involved in the EU FP7 Project PPDR-TC. His research interests include network softwarization, public safety networks, and safety-related V2X systems.

MAURIZIO CASONI (Senior Member, IEEE) received the M.S. degree (Hons.) and the Ph.D. degree in electrical engineering from the University of Bologna, Italy, in 1991 and 1995, respectively. In 1995, he was a Research Fellow with the Computer Science Department, Washington University at St. Louis, St. Louis, MO, USA. He is currently an Associate Professor of telecommunications with DIEF, UNIMORE, Italy. He has been responsible for the EU FP7 Projects E-SPONDER and PPDR-TC at UNIMORE.