Aggregating Without Bloating: Hard Times for TCP on Wi-Fi

Carlo Augusto Grazia, *Member, IEEE*, Natale Patriciello, Toke Høiland-Jørgensen, Martin Klapez, and Maurizio Casoni, *Senior Member, IEEE*

***Abstract*— Since the definition of the bufferbloat phenomenon, several Linux kernel modules have been introduced in its TCP/IP stack, and there is a lack of experimental studies on their effects when coupled with WLAN technologies, in particular, IEEE 802.11n and IEEE 802.11ac. One essential algorithm introduced is named TCP Small Queues (TSQ) and has the role of limiting the number of packets that a TCP socket can enqueue in the stack, waiting for the physical layer to send the packets before enqueueing extra data. A second significant TCP algorithm is named TCP Pacing (TP) and regulates the pace used by the socket to enqueue packets in the stack, regulating the formation of bursts of data. These mechanisms affect the frame aggregation logic on WLAN networks and compromise the throughput-latency tradeoff of all the TCP variants. This paper presents an experimental evaluation of these techniques investigating the wireless network performance of several TCP congestion control variants under the presence of different TSQ and TP policies, modeling also their interaction.**

***Index Terms*— Congestion control, frame aggregation, latency, pacing, TCP, TSQ, WLAN.**

I. Introduction

THE increase of Wi-Fi usage encourages research and development on new optimized standards, as well as refinements of the current ones [1]–[3]. For instance, IEEE 802.11n/ac/ax need frame aggregation to boost the overall throughput. Frame aggregation increases spectrum efficiency by sending more than one frame in a single transmission opportunity [4]. Simultaneously, the inflation of buffers along network paths intended to limit the loss rates of links led to the, now well recognized, bufferbloat problem [5], that could cause up to seconds of needless queuing delay.

In this paper, we take the Linux kernel as the networking reference point, as it is currently running the vast majority of Internet-connected servers for its robustness, security, and speed. Several solutions spanning the entire Linux networking stack have been recently introduced to optimize network performance. At the transport layer, the TCP congestion control domain has seen proposals such as Google BBR [6], TCP Small Queues (TSQ), and TCP Pacing (TP). The acronym BBR derives from "bottleneck bandwidth" and "round-trip propagation time"; indeed, the BBR TCP congestion control algorithm has been designed to make efficient use of the network channel while keeping latency under control; BBR does this by building a model of the network path in order to avoid and respond to actual congestion. TSQ is a process coupled with every TCP congestion control algorithm, designed to limit the number of packets that can be enqueued down the stack at any given time on the basis of the rate at which the Network Interface Card (NIC) effectively dispatches frames. TP is a third module that controls the pace at which the enqueue occurs. Together, TSQ and TP regulate how much data to enqueue and how fast it is enqueued, respectively. At the network layer, a hybrid packet scheduler and Active Queue Management (AQM) algorithm called FQ-CoDel [7] has been instead introduced to address the bufferbloat problem directly. FQ-CoDel prevents the formation of large buffers by preferentially dropping packets that remain in the queue for an excessive time.

TSQ and TP perform remarkably well over wired links, but they can affect the frame aggregation logic of wireless standards that use it. We analyzed this limitation in [8] for Cubic coupled with TSQ, but, to the best of our knowledge, literature is still lacking a precise analysis of the effects of TSQ and TP on TCP congestion controls like BBR and other TCP variants available in the Linux kernel. This paper studies these effects both individually and in conjunction over two wireless technologies: IEEE 802.11n and IEEE 802.11ac. These standards need frame aggregation to utilize the spectrum fully. Among those requiring frame aggregation, currently, they are also the most widely used. Our experiments show that the only way to exploit the Wi-Fi bandwidth in upload efficiently is to relax the limit imposed by TSQ and TP, allowing for the frame-aggregation formation and higher throughput. The solution is not a silver bullet and requires a fine-grained tuning according to the aggregation size used by the specific technology under investigation and the TCP variant used, which affects the throughput-latency tradeoff. We selected two specific values of TSQ, now included in the Linux kernel mainline for the Atheros drivers, for the IEEE 802.11n and IEEE 802.11ac technologies.

The rest of the paper is organized as follows: Section II describes the related work, while Section III presents the

Manuscript received August 7, 2019; revised January 30, 2021, August 27, 2021, December 7, 2021, and January 7, 2022; accepted April 24, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Guan. *(Corresponding author: Carlo Augusto Grazia.)*

Carlo Augusto Grazia, Martin Klapez, and Maurizio Casoni are with the Department of Engineering Enzo Ferrari, University of Modena and Reggio Emilia, 41125 Modena, Italy (e-mail: carloaugusto.grazia@unimore.it; martin.klapez@unimore.it; maurizio.casoni@unimore.it).

Natale Patriciello was with the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), 08860 Barcelona, Spain. He is now with Vonage, Video Media Platform Engineering, 08018 Barcelona, Spain (e-mail: npatriciello@acm.org).

Toke Høiland-Jørgensen is with Red Hat, Raleigh, NC 27606 USA (e-mail: toke@redhat.com).

Digital Object Identifier 10.1109/TNET.2022.3171594

current Linux TCP/IP stack. Section IV models the interaction with TSQ, TP, and frame aggregation mechanism. Section V describes the testbed used to produce the results analyzed in Section VI. Finally, Section VII concludes the paper.

II. Related Work

TCP congestion controls performance is always under a microscope. The literature contributions range over a variety of scenarios, with different topologies and technologies, addressing several issues. To give an example, in [9] Zhou *et al.* classified and analyzed stalls at the server-side dealing with a fundamental TCP characteristic: the Retransmission Time Out (RTO) value. RTO can indeed affect TCP performance in the wild. The authors realized the existence of this issue by observing several real-traffic TCP traces concluding the necessity to have tailored RTOs for different services. In another recent study [10], the authors proposed a new method for testing TCP congestion controls in a scalable way, leading to a list of bugs never reported before. Another challenge is to identify the TCP variant running on a network; for instance, authors in [11] provided a tool to identify the TCP congestion control used by a remote server.

TCP congestion controls performance is a well-studied topic in the literature. Many contributions cover the main aspects, while others focus on several niches. Some novel contributions are based on machine learning to train the congestion control algorithm in a large variety of networks. This is the case of [12]–[14], to cite some of them which are challenging the research on this topic comparing the performance with existing congestion controls. There are also contributions focusing on rethinking the TCP protocol entirely, deprecating its macroscopic model after the introduction of BBR [15]. However, to the best of our knowledge, there is a relevant part missing. That is a proper investigation of the TSQ and TP algorithms and an experimental review of novel TCP variants over WLANs. TSQ, TP, and other novel algorithms like BBR and FQ-CoDel, have been introduced to mitigate the bufferbloat phenomena. Still, the interaction with new Wi-Fi modules carries uninvestigated drawbacks.

We now present relevant works that get close to our contribution by following a bottom-up approach. A detailed analysis of the WLAN frame-aggregation logic has been provided through simulations in [16]. Moreover, a throughput comparison between the new IEEE 802.11ax and the current IEEE 802.11n/ac standards has been provided in [17] through the ns-3 Network Simulator. Another work based on simulations [18] analyzes the data rate of IEEE 802.11n/ac under power constraints. These are all examples of works close to the topic that miss the opportunity to investigate possible issues arising from TSQ and TP due to the absence of these mechanisms in the ns-3 simulation environment. Nevertheless, real tests involving TCP over Wi-Fi exist. It is the case of [19] that presents an experimental evaluation for the power-throughput tradeoff of IEEE 802.11n/ac technologies, including also smartphones in the testbed. Even papers related to Multi-Path TCP (MPTCP) are supported by experimental results [20]. However, the only available source revealing that TSQ breaks the frame-aggregation logic of IEEE 802.11n/ac is [8], which provides only data related to Cubic and does not include TP analysis. For what concerns TCP pacing, there is a decade of literature investigating the mechanism. This is the case of [21] where TP is studied in multi-hop ad-hoc wireless networks as well as [22] where the same mechanism is investigated for solutions tailored for data-centers networks.

Moving towards general evaluations of different TCP variants over wireless networks that do not deal with the frame-aggregation problem and the TSQ/TP algorithms, there are mainly works based on simulations [23], [24]. On the other side, for the wired technologies, it is easy to find works based on both simulations and real testbeds [25], [26]. Even the learning-based congestion controls are part of this group. PBE-CC [14] has been designed for cellular networks, where the environment is highly dynamic, but the implementation is a proof-of-concept unavailable in the Linux kernel for real-test comparison involving TSQ and TP modules. Same discussion for [13] which is based on emulated environments. Even Rein [12], one promising learning-based congestion control, has been implemented based on an old kernel version 4.14 never included in the mainline. Numerous scientific contributions have analyzed BBR performance with different technologies. For example, some recent works have tried to answer this question: “will TCP work in mmWave 5G cellular networks?”. One work [27] is based on simulation and also includes BBR between the TCP congestion controls investigated. A general issue for mmWave is that it is challenging to exploit the available bandwidth during “irregular” time intervals for the currently available TCP variants. Continuing on the cellular network topic, with the current 4G available technology, BBR has been tested in [28] and [29]; both the papers conclude that BBR outperforms NewReno and Cubic in terms of throughput and latency, while in some network conditions, BBR struggles in maintaining fairness between flows. Concerning fairness, when different RTT flows are in place, this work [30] shows that it is hard to achieve fairness between BBR and Cubic, while thanks to FQ-Codel [7] the unfairness gap can be reduced remarkably. A variant of BBR, called Modest-BBR [31], modifies BBR by reducing its aggressiveness and increasing fairness with Cubic while still maintaining similar performance to the original BBR. To conclude the picture, the following two works investigate the behavior of mixed BBR and Cubic traffic by dealing with the internal parameters of BBR, in particular with its cycle [32]. Considering real tests of BBR over Linux systems, in [33] BBR and Cubic have been tested over standard Gigabit Ethernet wired networks with a 4.9 kernel version. The paper shows that BBR does not meet its standard behavior when multiple flows are in place in terms of both 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 in [34] and, recently, on IEEE 802.11ac in [35]. Moving to the improvement of BBR performance, several scientific works tried to solve well-known

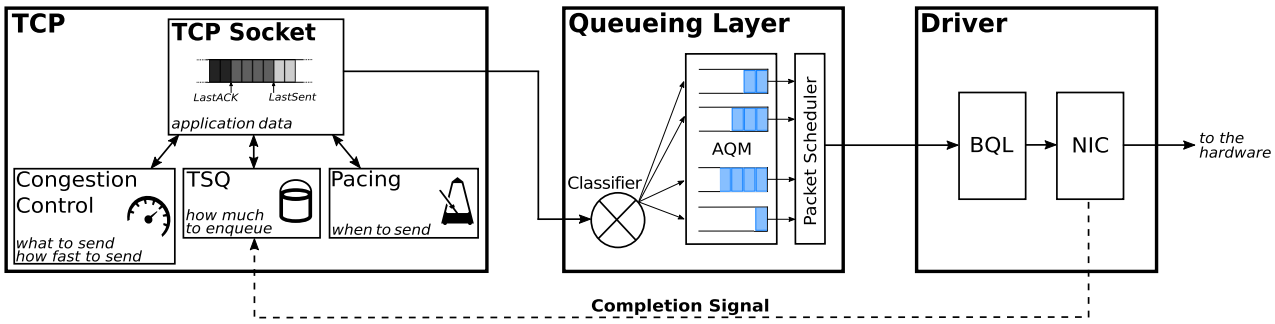


Fig. 1. Linux TCP sender architecture.

issues of BBR related to RTT fairness and also Wi-Fi inefficiencies. Concerning the latter, Google itself proposed a patch initially, through an RFC, named BBR-DEV [36], and then a new BBR version that became available since the Linux kernel version 5.x which is named BBR v2 [37]. BBR-DEV can operate increasing the BBR throughput when the TCP sender is ethernet-connected 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 also introduces an adaptive drain technique that has the goal of lowering queuing delays. BBR v2, instead, aims to take advantage of a tailored TCP pacing to increase frame aggregation and allow BBR to improve throughout in WLAN environments, regardless of the sender connectivity, either wired or wireless. We also proposed a BBR variant named BBRp [38], which customizes the BBR cycle dealing with the pacing gain and draining phase. The BBRp goal is to allow frame aggregation without excessively bloating the bottleneck queue, allowing for better performance in WLAN environments without compromising the BBR model-based nature in wired ones. The BBRp publication [38] includes tests in which the transmitting node is not directly connected to a Wi-Fi interface, and the Wi-Fi hop is just on the download path; this masks the TSQ impact, and almost all the TCP algorithm manifests similar results except for BBR v1, unable to aggregate properly due to pacing issues.

To summarize, on one side, there is the community effort to improve the performance of BBR over WLANs, where TP plays a critical role. On the other side, there are no works that focus on different TCP variants together with TSQ or TP, highlighting their interference. This manuscript aims to close the gap, focusing on the challenging WLAN scenario, which involves a frame aggregation mechanism.

III. Linux TCP/IP Stack

This section describes the current TCP/IP stack of the Linux kernel, including all the new parts covered in this paper, like TSQ and TP, as well as the Queueing Layer (QDisc) and the Driver blocks, all depicted in Figure 1. The current Linux TCP module is composed of three algorithms: TCP Congestion Control, TCP Small Queues, and TCP pacing. On top of this module, there is the TCP Socket, which generates the TCP segments and manages the ACKs. Every TCP connection is mapped with a specific TCP socket, and the packets are managed according to the three algorithms.

A. Congestion Control

It is a well-known part of the TCP module, rich in literature contributions about possible algorithms that can be used as congestion control. In this paper, we test Cubic [39], the current Linux default, BBR [6] (and three of its variants), designed by Google and incrementally deployed in many nodes, New Vegas [40], a timestamp-based congestion control like BBR, two window-based variants which are New Reno [41] and HighSpeed (other TCP variants like Westwood+, Reno, Bic, Illinois and Scalable, available in [42], are not included here for space constraints), and a window-rate hybrid TCP variant named YeAH [43]. These algorithms are well different in terms of approach: BBR and New Vegas are rate-based variants in which the concept of time is stressed to reduce latency as the main goal, while Cubic and the others are mainly window-based and have goodput maximization as the primary goal. Each congestion control is responsible for fundamental operations like the computation of the sending rate and the congestion window size (CWND), as well as the computation of the TCP parameters in the presence of congestion events or packet loss.

B. TCP Small Queues (TSQ)

It is an algorithm introduced by Google to mitigate TCP flows latency. The TSQ algorithm allows each TCP socket to enqueue a limited number of packets in its node stack. The goal is to mitigate the Bufferbloat [5] phenomena by avoiding the accumulation of packets in the sender node queues; the TCP socket is informed and allowed to enqueue a new packet in the stack only when the NIC finalizes the dispatch of a packet. The standard TSQ algorithm allows each TCP socket to enqueue a specific number of packets equivalent to the number of packets corresponding to 1 ms of latency at the current sending rate; this mechanism introduces an upper bound to the sending node queueing delay as a function of the flow throughput. This global limit of 1 ms has been proved in [8] to be too strict in a Wi-Fi environment where frame aggregation is not efficient with such a limit.

C. TCP Pacing (TP)

The algorithm defines the pace used to push the packets from the TCP module to the lower layers of the stack. While TSQ limits the number of packets enqueued, TP limits the internal rate for moving packets to the networking layer,

Algorithm 1 TCP Pacing Rate.

Input: TCP_SOCKET sk, **int** base_{RTT};

```
1: int rate = mss * sk→cwnd / base_RTT;
2: if sk→cwnd < sk→ssthresh / 2 then
3:    rate *= tp_ss_ratio;                    // SlowStart phase
4: else
5:    rate *= tp_ca_ratio;                    // Cong.Avoid. phase
6: end if
```

Algorithm 2 TCP Small Queue.

Input: TCP_SOCKET sk;

```
1: int limit;
2: limit = max(2 * sk→pktsize, sk→tp_rate ≫ 10);
3: limit = min(limit, tcp_limit_output_bytes);
```

forcing a time interval between a packet enqueued and another. Both TSQ and TP can help to avoid the formation of bursts, thus mitigating the Bufferbloat effect. TP's essence is to spread packets in the time domain by using the base RTT computed by the congestion control. Indeed, an essential parameter of the pacing algorithm is the base RTT of the network. Almost all the TCP congestion controls use a standard TP algorithm; BBR, which implements its pacing mechanism, is an exception. The standard Linux TP algorithm uses two default rates for pacing, and both are expressed as a percentage of the current rate of a TCP flow: 200% and 120%. The former is used during the slow-start phase, allowing to enqueue packets at a rate that is twice the current one, while the latter is used during the congestion avoidance phase, allowing to enqueue packets at a rate that is 20% higher than the current one. BBR uses a similar value called TP Gain, hardcoded in the BBR algorithm and not tunable in userspace, equal to a rate that is 25% higher than the current one.

D. TSQ and TP Interaction

The cooperative work of these two TCP submodules strongly impacts the way packets are delivered by the TCP socket. The two variables mentioned before for TP are `tp_ss_ratio` and `tp_ca_ratio`, used in the slow start and the congestion avoidance phases, respectively. The TCP socket's final TCP paced rate to deliver data is then adjusted with a `tp_ratio` that changes according to the TCP transmission phase, as shown in Algorithm 1. Converting the percentages described before, `tp_ss_ratio` is equal to 2 and `tp_ca_ratio` is equal to 1.2, respectively, allowing to probe for more bandwidth without forming excessive bursts of packets in the network queues.

On the other side, TP and TSQ define the number of packets that a TCP socket can enqueue in the sender stack. This quantity is a dynamic value, calculated according to Algorithm 2, equal to the amount of data that corresponds to a latency of 1 ms, by default. TSQ is also bounded between a minimum amount of packets (2 by default) and a maximum amount of bytes (128 KB by default). Algorithm 2 clarifies this behavior: the dynamic amount of data that can be enqueued is

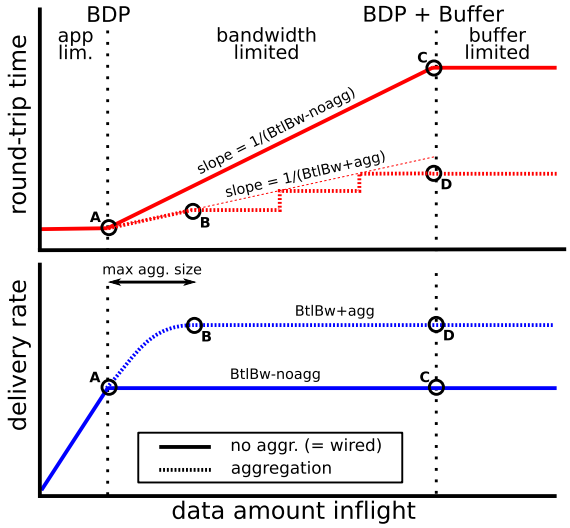


Fig. 2. Wireless bottleneck.

calculated through `tp_rate` $\gg 10$, which is a 10-bit shift of the current pacing rate that corresponds to a 1 ms latency. This mechanism helps the sender congestion control to mitigate the queueing delay inside the node, reducing the RTT.

In final, Figure 1 also reports the standard structure of the FQ-Codel [7] algorithm, the default option of many Linux distributions. Once the TCP socket forms a packet, the packet moves into the QDisc layer, where packet scheduling and AQM policies are applied to guarantee QoS constraints. When the packet is dequeued from the networking layer, it is delivered to the NIC driver, the piece of code that interacts with the hardware and delivers packets on the medium. Finally, a very last queue is present in the driver; it is typically a FIFO and is ruled by a Byte Queue Limit (BQL) [44], [45] to avoid excessive queueing.

IV. TSQ, TP and Aggregation Interference

The currently used model for representing bandwidth and RTT of a TCP traffic over a standard wired bottleneck, like the one employed by Google to design TCP BBR [6], is reported with solid lines in Figure 2. The model represents three regions: (i) the application limited one in which a TCP flow can increase the delivery rate without affecting the RTT, (ii) a bandwidth-limited one in which the TCP flow reaches the bandwidth-delay product (BDP) of the system and increasing the amount of data inflight has the sole effect to increase the RTT, since packets start to accumulate at the bottleneck link causing the so-called bufferbloat effect, and (iii) a buffer-limited region, where the bottleneck is full and starts to drop packets avoiding further RTT increments. For an additional description of the wired model, we refer to [6]. This model has been used to highlight the operating point of different TCP congestion controls with, as an example, the delay-based variant TCP New Vegas and the model-based variant BBR operating close to point `A`, and the loss-based variants like TCP Cubic and New Reno operating close to point `C`. The same model also holds for wireless bottlenecks that do not operate with frame aggregation, where each packet transmission is independent of the other.

Moving to Wi-Fi, in which packets are aggregated, we lose bandwidth and RTT linearity in the central region of Figure 2 and the optimal operating point becomes B, where frame-aggregation allows to reach the maximum throughput, while loss-based variants operate close to point D. The reason of the non-linear behavior is twofold: (i) the delivery rate is not constant as a function of the bottleneck queue length, and (ii) the increment of RTT is not merely the transmission delay of the k packets at the bottleneck queue. Further details on this effect can be found in [46]. Concluding, our goal is to model the delivery rate and the RTT as a function of the aggregate size, which depends on TSQ and TP rate.

We assume a single active station that transmits aggregates of k packets at a constant bitrate r, which is our system's bottleneck. In practice, the aggregates are composed of k packets, including frame overhead and padding, and the aggregate length is $k \cdot l \cdot 8 + l_{oh}$ bits, where l is the packet size in bits, and l_{oh} is the physical layer overhead. Moreover, focusing on the upload, the bottleneck link is directly the NIC of the sender, allowing us to model the number of packets at the NIC waiting to be transmitted as a function of the TSQ size ($\bar{t}$) and TP ratio ($\bar{p}$):

$$k(\bar{t}, \bar{p}) = \bar{t} \cdot \bar{p} \cdot c. \tag{1}$$

To simplify the discussion, we refer to a steady-state case in which the NIC queue is backlogged; this allows us to express $\bar{t}$ as a static quantity expressed in packets, and $\bar{p}$ as a multiplicative factor. This multiplication is indeed the cascade execution of Algorithms 1 and 2. The parameter c is a constant value, always in the $[0, 1]$ range, that represents which percentage of the product $\bar{t} \cdot \bar{p}$ is actually backlogged at the NIC. This parameter is essential to fit the model into the use-cases, where sender nodes may have different hardware or software characteristics. While the TCP socket has an imposed limit of $\bar{t} \cdot \bar{p}$ packets to enqueue in the stack, this does not mean that these packets will all be backlogged at the NIC and ready to form an aggregation. Depending on the load of the CPU usage or the number of active flows, some packets can still be waiting at the networking layers due to software bottleneck or high congestion, respectively. The parameter c includes these details focusing on the amount of the $\bar{t} \cdot \bar{p}$ product which is actually enqueued in the NIC, and so ready to be part of an aggregate. The time needed to transmit an aggregate of $k(\bar{t}, \bar{p})$ packets is:

$$TxTime(k(\bar{t}, \bar{p})) = \frac{k(\bar{t}, \bar{p}) \cdot l \cdot 8 + l_{oh}}{r} + t_{oh} \tag{2}$$

where t_{oh} is the per-transmission overhead, which encapsulates the inter-frame spacing, the average block acknowledgment time, and the average back-off time before transmission. A detailed explanation of t_{oh} overhead is given in [47]. Starting from this, we can compute the expected effective throughput $Thr(k(\bar{t}, \bar{p}))$, assuming no errors or collisions, and no other active stations:

$$Thr(\mathrm{k}(\bar{t}, \bar{p})) = \frac{k(\bar{t}, \bar{p}) \cdot l \cdot 8}{TxTime(k(\bar{t}, \bar{p}))} \tag{3}$$

If the number of packets $k(\bar{t}, \bar{p})$ enqueued in the ath driver is smaller than the *maximum aggregation size* (*mxg*), then the driver will transmit all the $k(\bar{t}, \bar{p})$ packets in a single aggregate. The time needed to complete the transmission is TxTime(k($\bar{t}$,$\bar{p}$)), under the steady-state hypothesis. Unfortunately, the amount of time TxTime(k($\bar{t}$,$\bar{p}$)) corresponds to the delay of only the last packet p_k, which is started to be transmitted with the aggregate as soon as it arrives in the queue. Instead, the first packet p_1, which will experience the same transmission delay of p_k since they belong to the same aggregate, has also waited in the queue for, at least, the arrival of the other packets, waiting for the aggregate formation [48]. This effect leads to a delay of, at least, 2· TxTime(k($\bar{t}$,$\bar{p}$)), since in [46] is demonstrated that the queueing delay, for the aggregate formation, is equal to the transmission delay under our hypothesis.

We consider the RTT experienced by our system as the sum of the RTT_{base} of the network with the contribution imposed by forming the bottleneck queue through queueing delay and transmission delay, considering ACKs delay negligible. The maximum $RTT(k(\bar{t}, \bar{p}))$ experienced in a queue of $k(\bar{t}, \bar{p})$ packets, with maximum aggregation size *mxg*, is:

$$RTT(k(\bar{t}, \bar{p})) = \begin{cases} RTT_{base} + 2 \cdot TxTime(k(\bar{t}, \bar{p})) & \text{if } k(\bar{t}, \bar{p}) \leq mxg \\ RTT_{base} + \left\lceil \frac{k(\bar{t}, \bar{p})}{m} xg \right\rceil TxTime(mxg) & \text{otherwise.} \end{cases} \tag{4}$$

indeed, if $k(\bar{t}, \bar{p}) > mxg$, there will be $\left\lfloor \frac{k(\bar{t}, \bar{p})}{mxg} \right\rfloor$ integer aggregates with *TxTime(mxg)* packets, and the queueing delay of the residual packets waiting to be transmitted. While the integer aggregates are transmitted, the residual packets in the queue will be grouped with the new arriving packets to form an aggregate of *mxg* size, which has a queueing delay equal to *TxTime(mxg)* [46]. Consequently, the RTT experienced by $k(\bar{t}, \bar{p}) >$ *mxg* packets became $RTT_{base} + \left\lceil \frac{k}{mxg} \right\rceil$ *TxTime(mxg)*. Similarly, we define the Wi-Fi bottleneck bandwidth as:

$$BW(k) = \begin{cases} Thr(k(\bar{t}, \bar{p})) & \text{if } k(\bar{t}, \bar{p}) \leq mxg \\ Thr(mxg) & \text{otherwise.} \end{cases} \tag{5}$$

Concluding the model description, it is important to notice that imposing a static maximum aggregation size equal to on packet (*mxg* = 1) we fallback to a wireless interface without frame-aggregation mechanism enabled. This is the case of the solid lines in Figure 2, obtained with this technique. For further details, the model without frame-aggregation has been investigated in [46].

V. TESTBED

This section describes our testbed, which is depicted in Figure 3. Each test involves a client, a server, and the access point that provides connectivity to the former two through a Wi-Fi and an Ethernet connection, respectively. All nodes run the Arch Linux distribution with a 5.4 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. The rest of the network is wired with Gigabit Ethernet interfaces, so they

TABLE I
MODEL PARAMETERS

Notation	Description	Value
l	packet size	1500 bytes
l_{oh}	physical overhead	48 bytes
r	station bit-rate: `AR9580 ath9k_htc`	216 Mbit/s
mxg	max agg. size: `ms` at the current rate	4ms
t_{oh}	channel access overhead	0.5 ms
RTT_{base}	base network RTT	2.5 ms
$\bar{t}$	TSQ size	[1, 400] pkts
$\bar{p}$	TP Ratio	1.2 and 1.6
c	Corrective backlog factor	0.8

TABLE II
TESTBED PARAMETERS

parameter	value
Kernel version	5.4-lts
Linux Distribution	Arch Linux
TCP Congestion Control	Cubic, BBR, New Vegas, YeAH, New Reno, HighSpeed, BBR-DEV, BBR v2, BBRp
TSQ type	`TSQ`, `2TSQ`, `4TSQ` `8TSQ`, `16TSQ`, `32TSQ`
TP Rate	`1p` (standard), `2p`, `3p`
QDisc	FQ_Codel
Wired links	1 Gbit Ethernet
Wireless Chipsets	Atheros AR9271 1x1, AR9580 3x3 MIMO Atheros AR5BHB116 2x2 MIMO Qualcomm QCA6178 2x2, 9880v2 3x3 MIMO
Wireless Drivers and Channels	`ath9k`: IEEE 802.11n 2.4 GHz (ch3) 40 MHz `ath9k_htc`: USB IEEE 802.11n `ath10k`: IEEE 802.11ac 5 GHz (ch58) 80 MHz
Tests	1-8 TCP Uploads, RRUL, UDP flooding
Metrics	TCP Throughput, TCP RTT, ICMP Latency, Frame aggr. size

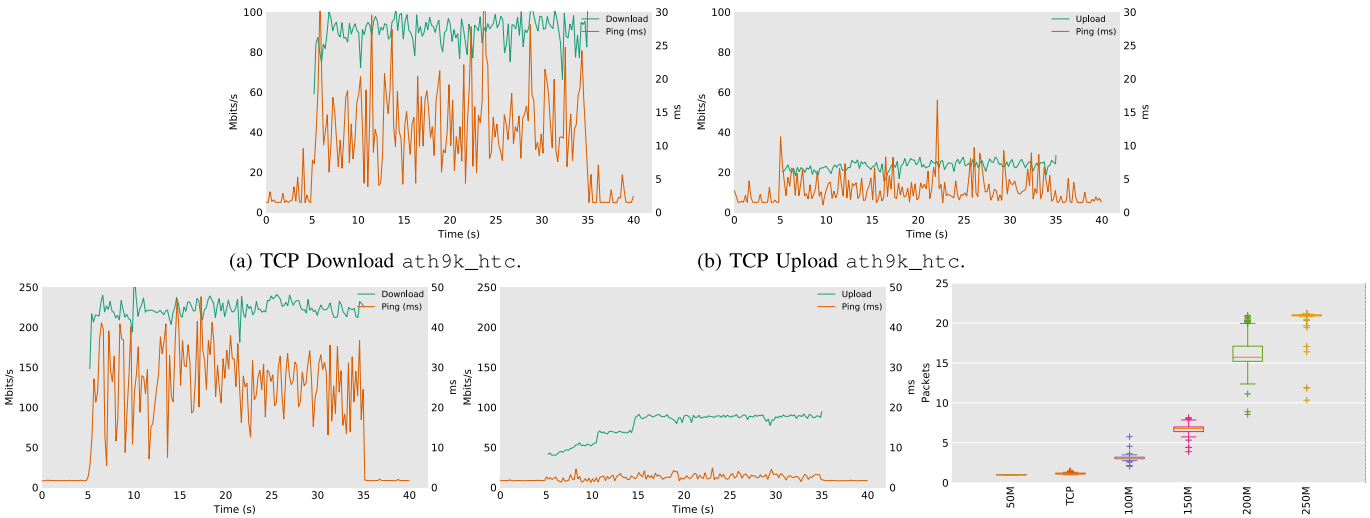
do not affect our tests since they are not the bottleneck. The wireless connectivity is given by PCIe Atheros chipsets supported by the ath9k and ath10k open drivers. The client uses different TCP congestion control algorithms (reported in Section III and Table II) and can set different possible TSQ limits [8] and TP rates. To overcome the inflexible standard behavior of TSQ, we patched the kernel to expose the TSQ core parameters and make it possible to disable or tune the TSQ logic or even impose static TSQ sizes expressed in bytes or packets. We name this solution Controlled TSQ (CoTSQ) [42]. While standard TSQ allows each socket to enqueue "1 ms of data" at the current rate, the CoTSQ patch allows changing the amount of data that can be enqueued at the current rate on the basis of additional time-windows. This limits the amount of data in the stack as a function of the `ms` parameter, resulting in a dynamic constraint, i.e., autotuning the number of bytes to enqueue as a function of the current rate. In this paper, we use values of 1 (standard TSQ), 2, 4, 8, 16, and 32 ms, because, at the kernel level, the TSQ size is managed as a bits shift operation (Algorithm 2) and power of 2 integers are preferred. The other most critical parameter introduced and tested in this paper is the TP rate. Since BBR does not react to any modification to the current Linux systems' standard pacing value, we patched BBR itself, exposing the TP Gain variable used internally. We named this patched version BBR+ and selected three possible pacing rates. These have been used for both the standard algorithm, as well as for BBR+. These pacing rates are named `1p`, `2p` and `3p`: where `1p` represents the standard pacing rates used by all the TCP variants, `2p` doubles the values and so on. Details about our BBR+ patch can be found in [42]. We followed the best practice document [49] provided by the bufferbloat community to configure our test computers and avoid the most common testing pitfalls. We then disabled all hardware offload features, turning them off (e.g., TSO/GSO). All of these adjustments serve to reduce sources of delay other than those induced by the algorithms themselves. We organized all the experiments reported in this paper by using the Flent [50] tool, which is a flexible and open network tester that allows managing different traffic typologies as well as auto-collect many performance results. Every test is organized as follows. A standard TCP flow runs in upload from the wireless client to the server. Each test runs for 40 seconds: the five initial seconds run with only ICMP traffic, the 30 middle seconds run with both ICMP traffic and the actual TCP transmission, and the five final seconds run, again, with only the ICMP traffic. This pattern enables to highlight the impact of the TCP traffic on the `ping` RTT, together with many other parameters related to the TCP traffic itself, e.g., throughput, TCP RTT, and CWND size. The parameters used to configure our experiments are reported in Table II. The testbed available in Figure 3 is simple yet effective since it represents the worst-case scenario for a single TCP upload on a Wi-Fi bottleneck, where frame-aggregation is used to maximize the available throughput. Different scenarios are available in our repository [42]; anyway, enriching the testbed has the sole effect of facilitating the TCP job. As an example, adding other clients would help TCP since the available bandwidth per node is reduced. The same consideration can be done moving the Server; indeed, both increasing the RTT or migrating to a remote bottleneck would reduce the gap toward the optimal throughput. Moreover, moving the bottleneck to the wired side, by reducing the ethernet bandwidth or by tuning the queueing disciplines, would equalize the performance of different TCP algorithms with respect to TP and TSQ, since the Wi-Fi frame-aggregation would not be involved.

Fig. 3. Physical testbed.

VI. RESULTS

A. Behind the Work

We start the numerical results section by showing evidence of the TCP upload inefficiency over IEEE 802.11n channels, under the premises exposed in Section III. In reference to Figure 3, we hereafter call *TCP upload* a TCP stream from Client `C` to Server `S` and *TCP download* a TCP stream from `S` to `C`. Figures 4a and 4b report the results of a Cubic download and upload, respectively, obtained using simple USB dongles with Atheros AR9271 1×1 MIMO chipsets and `ath9k_htc` as the driver. These are cheap devices that can



(a) TCP Download `ath9k_htc`. (b) TCP Upload `ath9k_htc`.

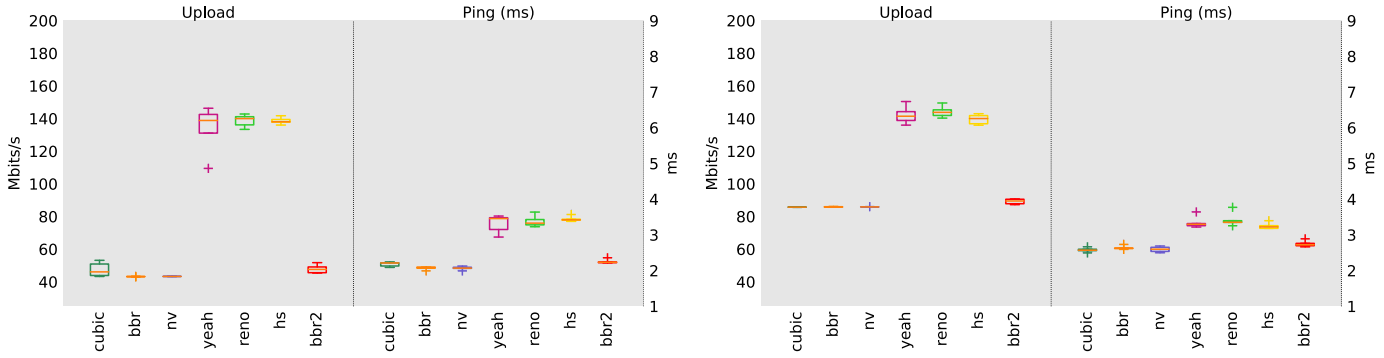
(c) TCP Download `ath9k`. (d) TCP Upload `ath9k`. (e) Frame Aggregation `ath9k`.

Fig. 4. One TCP flow in download vs. upload: TCP cubic.

be used to easily add wireless connectivity to any Linux computer since the Atheros drivers are supported by default. It is possible to notice immediately that, regardless of the hardware's relatively low capacity, there is a considerable discrepancy between the download and the upload stream in terms of TCP goodput (measured in Mbit/s). The Wi-Fi bitrate of the dongles is 150 Mbit/s, leading to a maximum TCP goodput close to 100 Mbit/s. The TCP download goodput reaches optimal and almost stable values close to 95 Mbit/s, while the TCP upload cannot reach goodput values higher than 30 Mbit/s. To clearly understand the reason for this unbalance, we repeated the same experiment with PCIe Atheros AR9580 3×3 MIMO devices with the driver `ath9k`. Other than the hardware being different, this also enables the possibility to capture Wi-Fi statistics like the frame aggregation size. Wi-Fi statistics data can be indeed collected only with `ath9k`, while `ath9k_htc` and `ath10k` perform rate control operations in firmware and do not allow to collect these results. Figures 4c and 4d reveal that changing the hardware does not lead to different conclusions, although in both cases is possible to reach higher goodput values thanks to the higher maximum bitrate of the 3×3 devices equal to 300 Mbit/s. The same unbalance persists, as the maximum TCP goodput reached in the download is higher than 200 Mbit/s, while the TCP goodput reached in upload struggles in reaching a value close to 100 Mbit/s. To investigate this phenomenon, we tried different `iperf` instances in upload by using UDP instead of TCP and incrementally increasing the bandwidth load from 50 up to 250 Mbit/s. The outcome has been that UDP can reach values higher than 200 Mbit/s in both download and upload. This enhances a gap between a single TCP upload goodput, and a single UDP upload one. To complete the picture, we then collected Wi-Fi statistics of both TCP and UDP experiments. Figure 4e reports the average frame aggregation size of the different UDP instances as a function of the throughput load and of the sole TCP upload that, as shown in Figure 4d, reaches goodput values between 50 and 100 Mbit/s. The result is clear and shows that while an increasing UDP load corresponds to an increased frame aggregation with a consequent goodput increment, the aggregation almost does not happen at all with TCP.

Hereafter, we refer with `ath9k` to the tests conducted with PCIe Atheros AR5BHB116 2×2 MIMO devices (maximum TCP goodput of 200 Mbit/s) and with `ath10k` to the tests conducted with PCIe Qualcomm QCA6178 2×2 MIMO devices (maximum TCP goodput of 400 Mbit/s).

We continue our analysis by altering the TCP congestion control. We report the results of 7 selected variants, namely Cubic, BBR, New Vegas, YeAH, New Reno, HighSpeed, and BBR v2. As mentioned in Section III, these are the most representative group of TCPs for our tests; we also tested all the other Linux default variants, and the results can be found in [42]. Figure 5a shows that none of the TCP variants are able to reach the optimal TCP upload goodput of 200 Mbit/s. In particular, BBR, BBR v2, and New Vegas perform very similarly to Cubic with TCP goodput values close to 50 Mbit/s and a latency close to 2 ms. The congestion-based variants register a surprising result: YeAH, New Reno, and HighSpeed are able to reach 140 Mbit/s of TCP goodput in upload by paying the price of a higher latency of 3.5 ms. Although TCP Cubic is a congestion-based algorithm like TCP New Reno, its throughput result is lower since different TCP slow-start algorithms are used. Such a result is different from our previous publication [38] (Figure 6a), where the performance of the congestion-based TCPs is equalized by disabling the TSQ logic. In other words, TCP Cubic manifests lower performance compared to TCP New Reno since the different slow-start algorithms interfere differently with the TSQ logic. Further details on the impact of the different TCP slow-start algorithms will be described in detail in Figure 12. Figure 5b, instead, reports the result of 4 simultaneously active TCP uploads. The first thing to notice is that the cumulative goodput increases by increasing the number of active TCP flows, in particular for the first three TCP variants of Cubic, BBR, BBR v2, and New Vegas. Increasing the number of TCP uploads has limited effects and does not change anything significant in terms of



(a) 1 TCP Upload. (b) 4 TCP Uploads.

Fig. 5. TCP upload: standard TSQ, Goodput vs. Ping, ath9k.

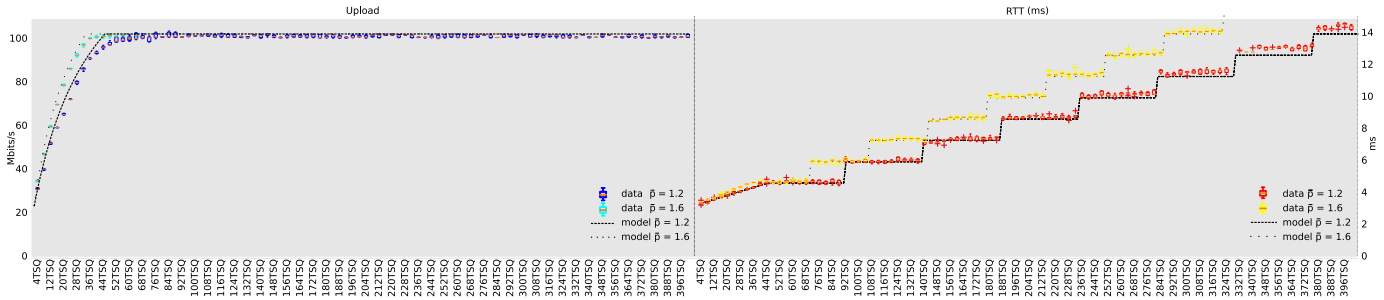


Fig. 6. Throughput and RTT on an `ath9k_htc` wireless bottleneck: data vs. model. $\bar{t} \in [1, 400]$ packets, $\bar{p} \in \{1.2, 1.6\}$.

goodput for the other variants. Considering that the optimal goodput value is around 200 Mbit/s, a clear inefficiency is present. The reason is related to the TSQ algorithm that, regardless of the TCP variant, breaks the frame aggregation mechanism, impeding the TCP algorithm to enqueue the number of packets needed to form larger frame aggregates to reach higher goodput. The presence of multiple TCP flows slightly improves the number of available packets; this is why Figure 5b registered slightly higher goodput values than Figure 5a.

B. Data vs. Model

To validate the model of Section IV, we run an experiment involving a single TCP upload with the `ath9k_htc` devices as a function of TSQ size $\bar{t}$ and TP ratio $\bar{p}$. We choose TCP Cubic for this test since its loss-based behavior allows to control the point `C` of Figure 2 by controlling the amount of NIC packets through $\bar{t}$, $\bar{p}$ and c, according to Equation 1. The parameter c has been calculated empirically through our test, fitting our sender characteristics for this experiment. Moreover, `ath9k_htc` is selected for the same reason; otherwise, with `ath9k`, the integrated FQ-CoDel in the driver would not allow us to increase the RTT over 5 ms as a function of the data inflight. The results of this experiment are reported in Figure 6; on the left, we correlate the throughput of the TCP Cubic flow with Equation 3 while, on the right, we correlate the RTT of the TPC flow with Equation 4. The model's curves have been computed using the parameters reported in Table I. It is possible to notice how the data collected strictly resembles the model of Section IV. We used the packet-size version of our TSQ patch for two reasons. First, it allows us to use fine-grained x-axes for the data collected (with the standard definition of TSQ based on the latency, the user can select only discrete values of `ms` following the power of 2). Second, it is more consistent with the model's steady-state assumption since the TSQ patch's ms-version is rate-dependent. The results show how both the initial ramp and the stairs behavior of the RTT present and how the TP rate impact as a multiplicative factor on the x-axes, increasing the amount of data inflight with the same TSQ value.

C. Impact of TSQ

We continue our analysis by using the ms-version of TSQ patch, which enables the tune of the TSQ size allowing each TCP variant to enqueue more than 1 ms of data at the current TCP rate. In particular, we allow to enqueue the equivalent of `x` ms of data, naming each test `xTSQ`, with `x` being an integer value. It is important to notice that this patch has been included in the Linux kernel mainline, and each Wi-Fi driver can now set the desired `xTSQ` value. The impact of different TSQ sizes on all the TCP variants is reported in Figure 7. We changed the TSQ size from the standard one of 1 ms, named simply `TSQ`, up to 16 ms, named `16TSQ`. The first thing to notice when looking at Figure 7 is that, with the exception of BBR, goodput significantly improves for all the other TCP variants reaching 200 Mbit/s in several instances. Another thing to notice is that, in general, the TCP goodput grows as a function of the TSQ size and, at the same time, an increment in TCP goodput corresponds to an increment of the `ping` registered latency. Among the TCP algorithms able to reach the optimal goodput of almost 200 Mbit/s, the best one in terms of goodput-latency tradeoff is YeAH, with 7 ms of latency compared to the 8 ms of latency of Cubic, New Reno, and HighSpeed. New Vegas exhibited an unusual behavior; it

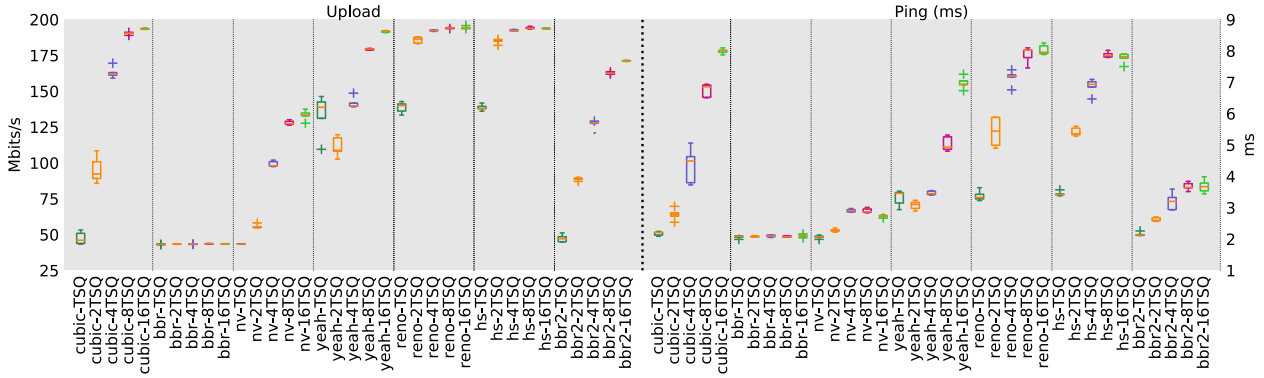
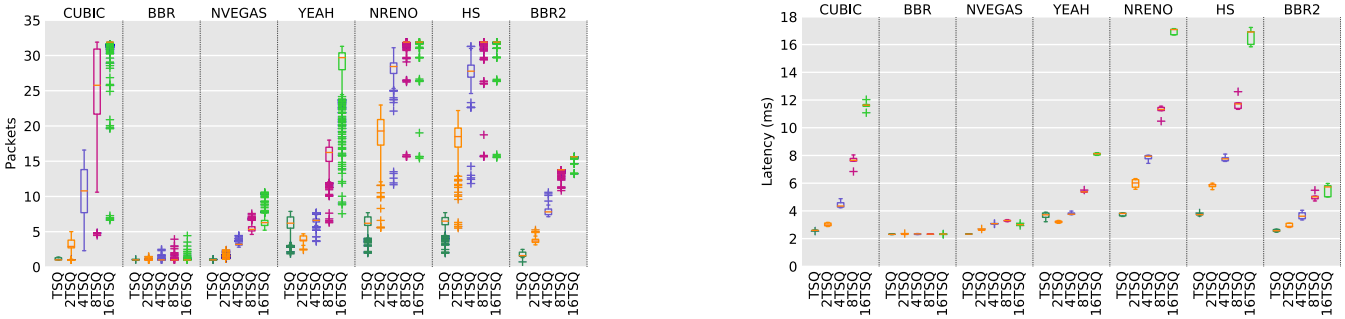


Fig. 7. One TCP flow in upload: different TCP & TSQ, Goodput vs. Ping, ath9k.



(a) Frame Aggregation.

(b) TCP RTT.

Fig. 8. One TCP flow in upload: Different TCP, frame aggregation and TCP RTT, ath9k.

responds to the TSQ increment by increasing the TCP goodput from less than 50Mbit/t to almost 150 Mbit/s, maintaining a latency stably below 3 ms. As already mentioned, the only exception here is represented by BBR, which does not respond to the TSQ variations due to its encoded behavior to control the latency (down at 2 ms, even relaxing the TSQ constraints) as a main figure of merit. On the other side, the new BBR v2 version can allow a throughput growth as a function of the TSQ size, reaching almost optimal values in between YeAH and New Vegas in terms of throughput, with optimal latency close to New Vegas.

To deeply understand the impact of the TSQ size increment, we refer now to Figure 8, where data regarding the frame aggregation size and the TCP RTT are reported from the same experiment of Figure 7. According to the discussion in Section IV, the optimal goodput is reached by the combinations of TCP and TSQ approaching the maximum aggregation size. Except for New Reno and HighSpeed (as well as for many not reported variants like Scalable, Illinois, Westwood+, and Hybla) that react very similarly to the TSQ variation, every other TCP variant manifests different behaviors. Compared to New Reno, Cubic aggregates less as a function of the TSQ registering lower RTT values. Figure 8b reveals something that is missing in Figure 7 and Figure 8a, i.e., the hidden difference between New Reno with `8TSQ` vs `16TSQ`. Despite the same aggregation size of 32, the same goodput of almost 200 Mbit/s, and the same `ping` latency of 8 ms, relaxing the TSQ constraints from `8TSQ` to `16TSQ` entails a rise of the TCP RTT from 12 to 18 ms. The same holds for HighSpeed. This because, according to the TSQ mechanism, more data is allowed to be pushed down in the stack, filling the buffers and causing a queueing delay that increases the TCP RTT measurements. Again, YeAH registered the best tradeoff between aggregation size and TCP RTT; it reaches an aggregation average of 30 packets, getting goodput values close to 200 Mbit/s (Figure 7), yet being able to contain the TCP RTT at 8 ms. The above considerations related to New Vegas apply again here. Figure 8a gives some more insights into the BBR behavior. The average BBR aggregation size is firmly stable at 1, and increasing the TSQ size causes only a few attempts to aggregate more (e.g., five packets with `16TSQ`). This likely happens during the BBR probe phases, while the effects are immediately corrected by the BBR drain phases, where the latency is locked as the most important parameter to control. The evolution of BBR, instead, confirm also with these figures of merit its good performance; BBR v2 allows for frame aggregation while still maintaining very low latencies, with a good tradeoff comparable to YeAH, with the sole remarkable difference at `16TSQ` where YeAH almost reaches optimal throughput while BBR v2 reaches 175 Mbit/s according to the previous results.

As a last consideration, Figure 8a let us explain the different TCP goodput values shown in Figure 5a. When standard TSQ is in place, New Reno, YeAH, and HighSpeed can aggregate between 6 and 7 packets, while Cubic, BBR, BBR v2, and New Vegas do not aggregate at all. The reason for this inequality lies in the way time is used in the TCP variant code and the interaction with the TP module that will be investigated in the following part of this section. The result is that New Reno, YeAH, and HighSpeed are more likely to enqueue bursts of packets simplifying the formation of

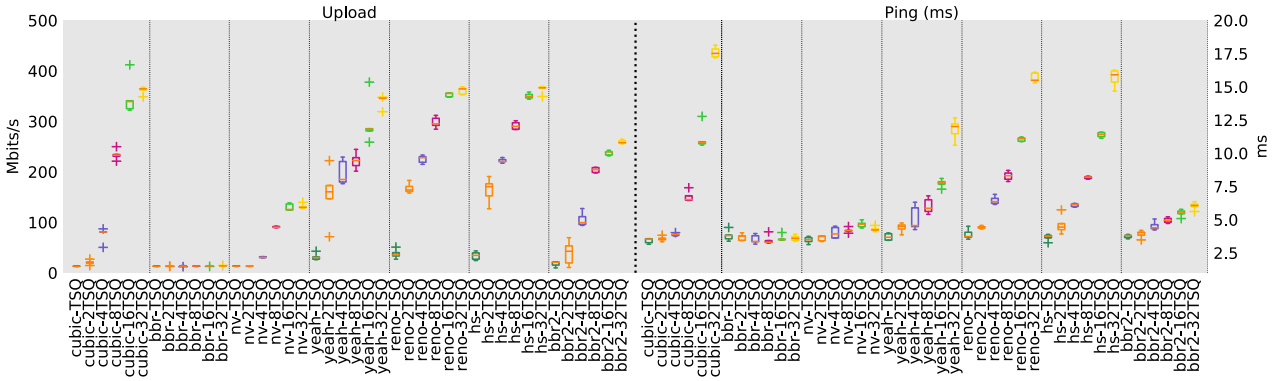


Fig. 9. One TCP flow in upload: different TCP & TSQ, Goodput vs. Ping, ath10k.

aggregates and consequently reach higher TCP goodput values with the standard TSQ. The drawback of these TCP variants is that the TCP RTT is slightly higher and close to 4 ms even with the standard TSQ in place.

D. IEEE 802.11ac

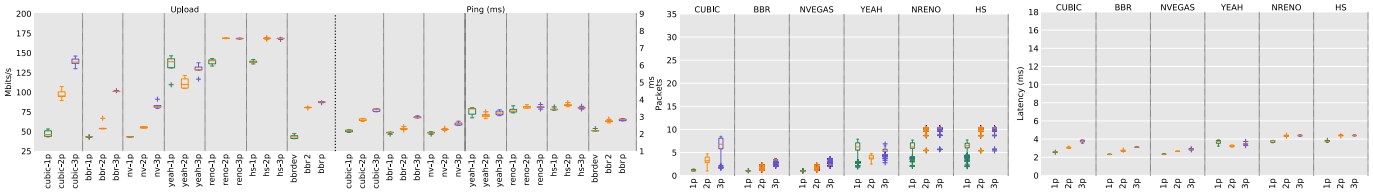
We now shift technology from IEEE 802.11n to IEEE 802.11ac, moving from the `ath9k` driver to the `ath10k` one. It is important to remark that with `ath10k` it is not possible to collect information related to frame aggregation as well as any Wi-Fi statistic in general, due to the closed firmware. Nevertheless, we are able to highlight the impact of the TSQ size on the TCP goodput and latency. Similarly to Figure 5, also in an IEEE 802.11ac environment, increasing the number of simultaneous TCP uploads from 1 to 4 does not solve the inefficiency imposed by standard TSQ size, the data related to the same test on `ath10k` driver are available in [42] and omitted here for space constraints.

The results related to the impact of the TSQ size during a single TCP upload are depicted in Figure 9. Here, due to the higher maximum bitrate allowed by IEEE 802.11ac, we also included the results of `32TSQ` in the plot. Indeed, optimal throughput with `ath10k` can be reached by allowing more data to be enqueued with respect to `ath9k`. The general trend is similar to the analogous experiment with IEEE 802.11n; HighSpeed and New Reno behave the same, but reaching 400 Mbit/s with the `16TSQ` and `32TSQ` configurations, increasing the latency from 11 ms to 16 ms when moving from the former to the latter. YeAH obtains similar results, although more constrained in terms of goodput and latency for the same TSQ sizes. New Vegas reacts to the TSQ size change with a weaker goodput increment that is up to 150 Mbit/s, paired to a negligible increment of latency. BBR continues to be unresponsive to the relaxed TSQ constraints. Similarly, BBR v2 poses itself in between New Vegas and YeAH in terms of throughput with slightly higher latency values with respect to New Vegas. Finally, Cubic get close to the optimal goodput with the configurations of `16TSQ` and `32TSQ`, while New Reno and HighSpeed show similar results. To conclude, comparing Figure 7 to Figure 9 and despite the similar trends, it is clear that IEEE 802.11ac needs to relax more the TSQ constraints to boost goodput to a level close to its maximum values, as these are higher than IEEE 802.11n.

E. Impact of TP

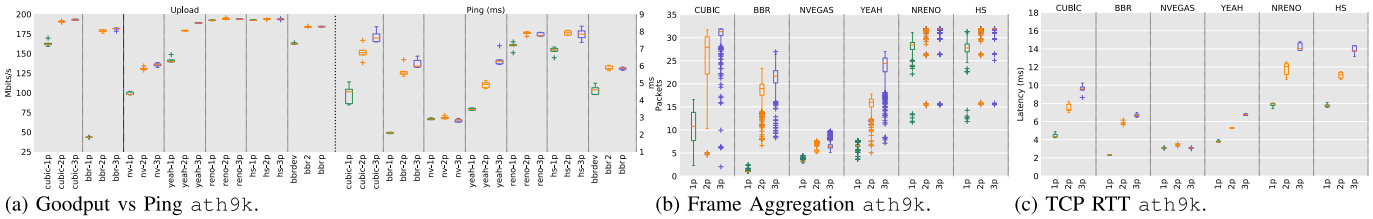
For this run of tests, we patched the BBR internal pacing module in order to make it respond to the same variations that we are going to impose on the other TCP variants, i.e., when we double the "global" pacing rate, we want the BBR pacing rate to double as well. Since the BBR variants of BBR-DEV, BBR v2 and BBRp deals with specific modification on the BBR pacing system, we include them here in the results with a more fair comparison.

With the following set of experiments, we aim to highlight TP's effect on the frame aggregation size and, consequently, on the tradeoff between goodput and latency. To isolate the pacing effect, with the `ath9k` configuration, we adopt the standard TSQ and transmit a single TCP upload. Figure 10 shows the results in terms of goodput vs. latency and frame aggregation size vs. TCP RTT, registered by varying the TP rate from the standard value identified with `1p`, to `2p` by doubling it for both the slow start and congestion control phases, and to `3p` by triplicating it in the same way. It is possible to notice that even if the standard TSQ limits the amount of TCP data that can be enqueued in the NIC, TP plays a fundamental role. Excluding for the moment YeAH, all the TCP variants can form larger frame aggregates as a function of the pacing rate, increasing the TCP goodput. This is because the higher the pacing rate, the higher the probability of forming a burst of packets instead of spreading them over time, therefore helping the formation of frame aggregates that result in a higher goodput. Cubic shifts from 45 Mbit/s at `1p` to 140 Mbit/s at `3p` paying 1.5 ms of extra latency and moving from 1 to 7 packets per aggregate. BBR and New Vegas behave very similarly, reacting with less but still significant effects to the pacing rate, starting to aggregate packets and doubling the goodput, from 40 Mbit/s to 100 Mbit/s, in the BBR case. New Reno and HighSpeed also improve as the TP parameter increases. Both saturate the channel with 175 Mbit/s by forming aggregates composed of 10 packets, already from a pacing rate equal to `2p`. YeAH, instead, manifests a counterintuitive degradation of performance. A possible explanation lies in the hybrid nature of YeAH. For what concerns the BBR variants, BBR-DEV provides similar results to standard BBR since it is unable to improve BBR perform it the first hop is wireless, which is the case of our upload experiment. BBR v2 and BBRp, instead, close the gap with the loss-based TCP variants, with



(a) Goodput vs Ping `ath9k`. (b) Frame Aggregation `ath9k`. (c) TCP RTT `ath9k`.

Fig. 10. Effect of TP: standard TSQ.



(a) Goodput vs Ping `ath9k`. (b) Frame Aggregation `ath9k`. (c) TCP RTT `ath9k`.

Fig. 11. Effect of TP: 4TSQ.

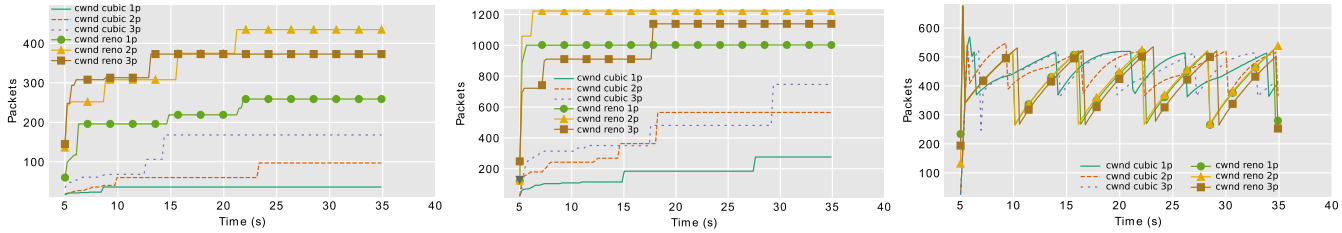
performance similar to Cubic coupled with a pacing rate of `2p`, both in terms of throughput and latency.

We now show the same experimental results where we relaxed the TSQ size at `4TSQ` in Figure 11, focusing again on the impact of different TP rates. We show the performance of `4TSQ` since it is the default TSQ size included in the Linux kernel for the `ath9k` driver. Other figures showing different TSQ sizes can be found in our repository [42], not included here for space constraints.

Cubic reaches the maximum aggregation size when TSQ is relaxed to 4 ms, as Figure 11 attests. Also, BBR and YeAH have a more significant response to the different TP rates with the `4TSQ` configuration, reaching almost the optimal goodput. The TP increment is fundamental to BBR for enabling the aggregation of packets and raising the goodput from less than 50 Mbit/s to values higher than 175 Mbit/s. New Vegas, instead, manifests almost negligible performance variations as a function of the TP rate. It is crucial to notice how the TCP RTT of New Reno and HighSpeed starts to increase significantly with just `4TSQ` if a high TP rate is applied, reaching 14 ms with `3p`. The BBR variants all react positively, relaxing the TSQ constraints. Indeed, they all increase the throughput without compromising the latency. BBR-DEV improves the performance of BBR with standard pacing, while BBR v2 and BBRp, again, register almost optimal throughput values, close to the one obtained by BBR with high pacing rates. This result confirms how both BBR v2 and BBRp take advantage of higher customized pacing ratios, which help their respective algorithms probe for more available bandwidth once the TSQ allows it.

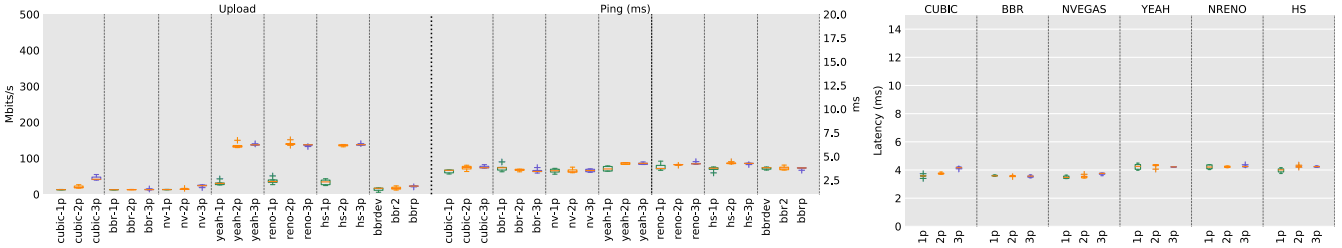
An important outcome of Figures 10 and 11 is the different impact of TP on TCP Cubic, on one side, and TCP New Reno and HighSpeed on the other side, all loss-based variants. To understand this difference, in particular the one visible in Figure 10, we show in Figure 12 a comparison between the CWNDs of Cubic and New Reno, at a function of the pacing ratios, in three different scenarios: standard TSQ in Figure 12a, `4TSQ` in Figure 12b and with the TSQ mechanism disabled in Figure 12c, respectively. This comparison is important since TSQ masks the TCP congestion control algorithm behavior when the bottleneck is local. In other words, if packets are accumulated in the sender NIC, like in our upload experiments, the CWND is then limited by TSQ, through the cross-layering interrupts, and not by the classic TCP congestion control algorithm. This mechanism is visible in Figures 12a and 12b, with long periods of several seconds in which the CWND is not updated. Such interference of TSQ with the CWND must be coupled with TP, since we have seen in Section III how TSQ and TP interact together to define the number of packets that can be enqueued, limiting the CWND. TCP Cubic behaves differently from TCP New Reno and, in general, from any other loss-based variant, since it implements a different slow-start algorithm, named Hybrid Slow-Start, and the interference of TSQ when the bottleneck is local forces TCP Cubic to switch to the congestion avoidance phase before, with respect to New Reno. The effect of this mechanism is twofold. First, the rapid growth of CWND is available only for TCP New Reno at slow-start, thanks to the fact that the pacing rate in the slow-start phase is higher. Second, it takes more time for TCP Cubic to increase the CWND in the congestion avoidance phase since the TP rate is lower. Anyway, the effect of increasing the TP rate is visible from the two figures, and the gap between Cubic and New Reno is reduced, increasing the TSQ size at 4 ms. The drawback imposed by the hybrid slow-start compromise a bit short-lived TCP streams. To conclude this analysis, we also report the results of the same experiment with the TSQ mechanism disabled, which can be done only with our patch, which are depicted in Figures 12c; this Figure shows that, by disabling the TSQ mechanism, the CWND is controlled again by the TCP congestion control algorithm, with the Cubic and New Reno shapes well identifiable and not affected by TP changes, since TP can only affect the TCP congestion control behavior through the TSQ mechanisms, according to Algorithms 1 and 2.

The impact of TCP pacing has been then tested over IEEE 802.11ac with the `ath10k` driver. Even in this case, TP affects performance with a remarkable impact in several contexts. Unfortunately, as stated before, we cannot report data about frame aggregation due to the driver limitation and, therefore, we report only the TCP goodput, `ping` latency, and TCP RTT.



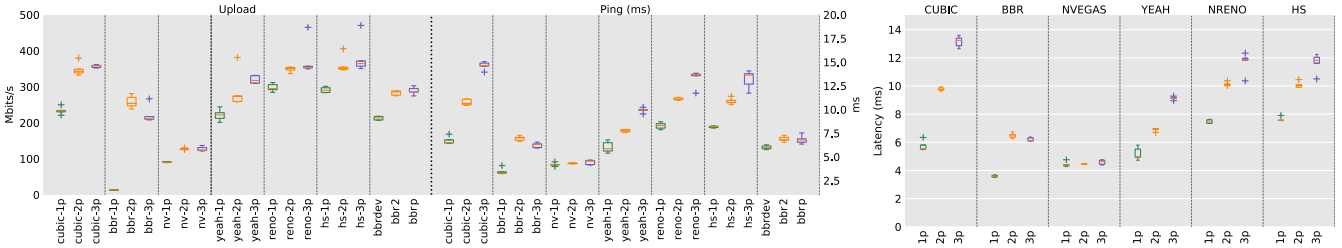
(a) Standard TSQ, ath9k. (b) 4TSQ, ath9k. (c) TSQ Disabled, ath9k.

Fig. 12. Cubic vs. New Reno: effect of TP on TCP congestion window.



(a) Goodput vs Ping ath10k. (b) TCP RTT ath10k.

Fig. 13. Effect of TP: standard TSQ.

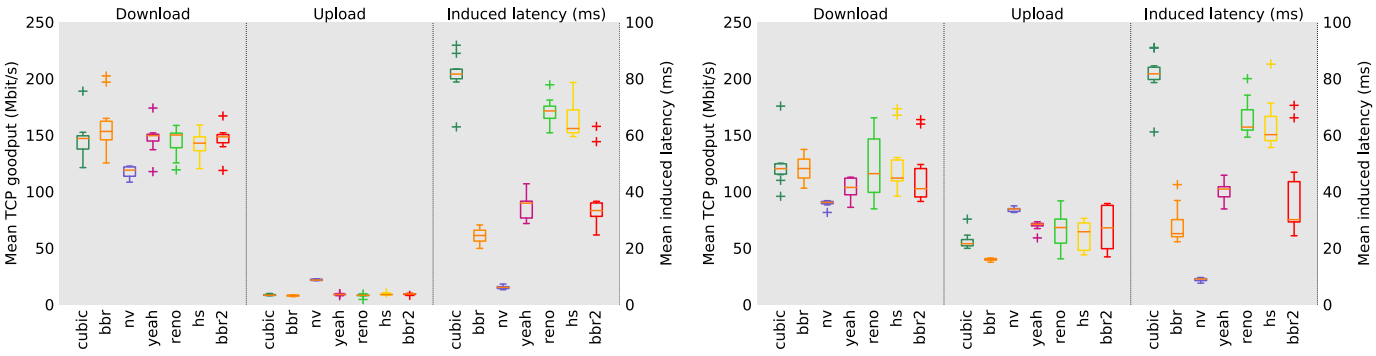


(a) Goodput vs Ping ath10k. (b) TCP RTT ath10k.

Fig. 14. Effect of TP: 8TSQ.

Moreover, starting from Figure 13 where we tested TP with standard TSQ, we pushed queue sizes up to 8TSQ, as shown in Figure 14, because the IEEE 802.11ac standard requires to relax more the TSQ size to appreciate optimal performance enhancements and 8TSQ is the default value imposed by ath10k driver in the current Linux kernel thanks to our patch. We demand to [42] the readers interested in the combination of different TP ratios with different TSQ sizes, where the extra plots are available. With standard TSQ, it is possible to notice in Figure 13 how increasing the pacing rate has an impact on some TCP variants. YeAH, New Reno and HighSpeed react similarly, moving from 30 to 150 Mbit/s of goodput when changing the pacing rate from 1p to either 2p or 3p, without significative latency or TCP RTT increments. Even Cubic increases a little its goodput from 25 to 60 Mbit/s, when paired with a pacing rate of 3p. BBR, all the BBR variants, and New Vegas, in this case, differently from the IEEE 802.11n environment, do not manifest any performance change dealing with pacing rate whilst standard TSQ are in place. In Figure 14, instead, the configuration 8TSQ is showed; Cubic, YeAH, New Reno, and HighSpeed all approach the optimal goodput of 400 Mbit/s. In this case, for Cubic, New Reno, and HighSpeed, the distinction of goodput between 2p and 3p is less marked; the main consequence is the latency increment that with 3p reach levels between 12 and 13 ms. YeAH moves into a good trade-off interval of goodput and latency, with the former ranging between 200 and 350 Mbit/s and the latter that ranges between 5 and 9 ms, as a function of the TP rate. Vegas is not able to boost its goodput more than to 200 Mbit/s with no sensible distinction between 2p and 3p, while BRR approaches 300 Mbit/s. The other BBR variants take advantages of the TSQ relaxation from 1 to 8 ms: BBR-DEV equals the performance of Cubic with standard paging rate, BBR v2 and BBRp, instead, slightly overcome the performance of BBR with 2p pacing ratio in terms of throughput with an almost equal latency, confirming how these two BBR variants well perform in WLAN environments.

A curious detail can be observed moving from ath9k to ath10k: with ath9k there is a clear distinction between ping latency and TCP RTT, especially with a combination of large TSQ and high pacing rates. This cause TCP RTT to increase more than the ping latency. The same cannot be said for ath10k. With ath10k, the two different values of ping latency and TCP RTT are the same in all the tests. The reason lies in the queueing discipline adopted by the two drivers. Indeed, ath9k has an integrated FQ-CoDel at the driver level, enabling fine-grained QoS and adopting different queues for TCP packets and ICMP (ping) packets. This means that when many TCP packets are enqueued, the TCP RTT increases due to the queueing delay. In contrast, according to the scheduling policy, the ICMP packets are less affected due to the different queues. The same does not apply to ath10k, that has not



(a) Download and Upload goodput vs Ping `ath9k`: Standard TSQ.

(b) Download and Upload goodput vs Ping `ath9k`: 32TSQ.

Fig. 15. RRUL test: standard TSQ vs. 32TSQ.

FQ-CoDel integrated into the driver, so all the packets (TCP and ICMP) fall in the same queue bloating both the `ping` latency and TCP RTT as a function of the queueing delay.

F. Real-Time Response Under Load

To conclude our experimental evaluation, we report the outcomes of tests performed by following the Real-time Response Under Load (RRUL) test suite specification. RRUL is a well-known test suite created by the Bufferbloat community to analyze network performance under heavy workloads because, under such circumstances, bufferbloat and other networking problems such as congestion and packet drops are easily induced. The Flent package integrates the RRUL test by default; it consists of 8 bidirectional streams (4 TCP streams in download and 4 TCP streams in upload) that run against ICMP and UDP traffic. We configured the RRUL in order to have always one TCP variant active for both the upload and download streams.

Figure 15a reports the results of the average TCP goodput in download, the average TCP goodput in upload, and the latency calculated on the `ping` RTT with the `ath9k` driver and the standard TSQ in place. The tests have duration and replication parameters those used in the previous sections. The main finding of these first tests is that there is a remarkable unfairness between download streams and upload streams, with TCP goodput close to 150 Mbit/s in download but constrained to 10 Mbit/s in upload for all the TCP variants tested, with the exception of New Vegas. In fact, the latter is slightly fairer with 125 and 25 Mbit/s for TCP goodput in download and upload, respectively. What can be used to differentiate TCP variants is the latency, which manifests different results as a function of the congestion control algorithm used. Cubic has the highest latency of 80 ms, while New Reno and HighSpeed have high latencies over 60 ms. YeAH, BBR, and BBR v2 stay steadily under 35 ms. In terms of latency, the best TCP variant is New Vegas with less than 10 ms, confirming the well-known characteristic of this algorithm to be less aggressive, thus reducing the effects of congestion. One of the reasons for the unbalance between the TCP goodput in download and upload is the TSQ algorithm itself. In our testbed, the TCP download streams are generated by the server, which is wired connected to the router and does not have any constraint related to the low-level aggregation of packets, i.e., each TCP variant is able to reach the highest possible throughput without incurring in limitations imposed by the TSQ mechanism. On the other side, for what concerns the 4 TCP upload streams, each TCP variant is experiencing the issue of a constrained frame aggregation. This is due to the TSQ mechanism that hampers frame aggregation when the node is wirelessly uploading content to the router. We then relaxed the TSQ size up to an equivalent of 32 ms and reported the results in Figure 15b. An important thing to notice is that the global fairness between TCP goodput in download and upload improves significantly, with different results for each TCP variant relaxing the TSQ limit at 32 ms, a significant value if compared to the values that we had tested when the single TCP upload streams were in place. Almost all the TCP variants manifest an increment in the TCP upload goodput and a reduction of the TCP download goodput in the `32TSQ` configuration, with a stable unchanged global latency. Considering the entire picture provided by Figure 15, it is clear that the only TCP variant able to get close to optimal fairness between download and upload streams is New Vegas, with no difference between download and upload goodput. Furthermore, New Vegas is also the best in terms of latency, continuing to guarantee `ping` RTTs lower than 10 ms. TCP YeAH and BBR v2 are the following in terms of fairness, with a difference of circa 30 Mbit/s between download and upload goodput, while the worst variant is BBR that is unable to provide enough upload throughput under a standard pacing rate scenario.

Focusing on the last experiment with the `32TSQ` configuration, we also tested the impact of TP on the TCP upload streams by using standard, double and triple TP rates of `1p`, `2p`, and `3p`, respectively. The results are reported in Figure 16. For the RRUL experiment, the TP rate of the upload streams has a minor impact. The difference, in terms of goodput, between the TCP download and upload is reduced to a few Mbit/s by increasing the TP rate for all the TCP variants. The impact of TP on the `ping` latency is instead negligible, with tiny increments as a function of the TP rate. The only TCP variant that benefits from the pacing increment is BBR, especially considering the upload goodput, remarking the inefficiency of BBR in the upload path for a Wi-Fi station. Concluding, also in this scenario that involves different pacing ratios, we report BBRp and BBR v2, whit similar results also in the RRUL test, and we also report BBR-DEV, which shows its nature of boosting the goodput only if the source is

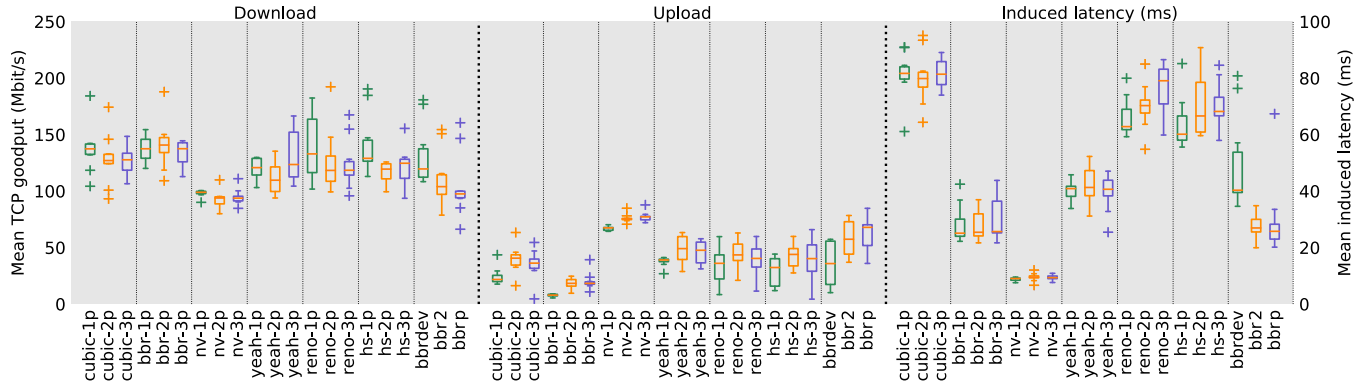


Fig. 16. RRUL test with 32TSQ: `ath9k` varying TP.

wired-connected, aggregating efficiently only in the downstream with respect to the upstream.

To summarize, the results obtained with our experiments can be discussed in terms of the tradeoff between throughput and latency, where TCP, TSQ, TP, and frame aggregation play a fundamental role. According to [15], the new direction for TCP macroscopic model design is the model-based world opened with BBR and, in particular, the new version BBR v2, which provides a remarkable balance between optimal throughput and optimal latency in almost all the experiments. The reason is that the only way to control the latency once the optimal throughput is reached is to monitor it. On the other side, latency cannot be statically imposed like what happened with standard TSQ, with only 1 ms allowed, in order to allow to boost the network efficiency when frame-aggregation is available. According to our results, the Linux kernel has been modified, introducing the possibility to refine the TSQ quantity as a function of the Wi-Fi driver characteristics. In particular, we selected 4TSQ and 8TSQ for the Atheros drivers used for IEEE 802.11n and IEEE 802.11ac technologies, respectively.

VII. Conclusion

The latest WiFi standards make extensive use of frame aggregation to reach higher speeds by increasing spectrum efficiency. On the other hand, higher layers have seen the introduction of techniques to keep latency under control while guaranteeing an increased utilization of the transmission channels. In this paper, we analyzed the impact that TSQ and TP may have on the network performance over IEEE 802.11n and IEEE 802.11ac channels, studying several figures of merit in conjunction with various TCP congestion control algorithms. Firstly, it has been clear that TSQ may have a strong negative impact on the frame aggregation techniques used by the aforementioned wireless standards by hampering it and resulting in a significantly reduced goodput. We developed and tested a patch that enables to tune the default TSQ size to address the issue, which is now included in the Linux kernel mainline. Extensive testings have been performed with various TSQ sizes in conjunction with different TCP congestion controls and over both wireless technologies. With the exception of BBR, all TCP variants respond positively to increased TSQ sizes at the cost of increased latencies, although with varying degrees of success. BBR does show some enhancements in the frame aggregation size, but they are limited and immediately hampered by the algorithm drain phases. The effects of different TP rates have also been isolated and analyzed, enhancing how imposing local limits on the amount of data that can be enqueued also impacts the congestion window size calculated by the TCP congestion controls. TSQ and TP interaction interferes with TCP congestion control algorithms when the bottleneck is local (e.g., the hybrid slow-start), and this research can help design new congestion controls that could take this into account. With the exceptions of BBR and YeAH, all TCP congestion controls allow forming larger frame aggregates as the pacing rate increases, resulting in higher goodput. As BBR embeds its own TP algorithm, we modified it to make it follow the system TP rate. With this change, BBR shows similar results to New Vegas, reacting with limited but significant positive effects to a pacing rate increase. Variations in TSQ sizes have then been studied in conjunction with variations in TP rates. Results indicate that they allow increasing frame aggregation in most circumstances, consequently also increasing goodput and latency. For most applications, the rise of the latter is still kept under acceptable levels. Finally, tests from the RRUL test suite have been performed to assess the different congestion controls fairness versus diverse TSQ sizes and TP rates, under the fundamental basis of having wired and wireless connections for download and upload transfers, respectively. While the fairness unbalances is in favor of wired connections, as expected, findings indicate that variations in TSQ size may improve it significantly, while deviations in TP rates provide almost-negligible enhancements.

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. Mag.*, 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, 2017.

[7] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. (2018). *FlowQueue-CoDel*. [Online]. Available: https://tools.ietf.org/html/rfc8290

[8] C. A. Grazia, N. Patriciello, T. Høiland-Jørgensen, M. Klapez, M. Casoni, and J. Mangues-Bafalluy, "Adapting TCP small queues for IEEE 802.11 networks," in *Proc. IEEE Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2018, pp. 1–6.

[9] J. Zhou *et al.*, "TCP stalls at the server side: Measurement and mitigation," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 272–287, Feb. 2019.

[10] W. Sun, L. Xu, and S. Elbaum, "Scalably testing congestion control algorithms of real-world TCP implementations," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[11] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1311–1324, Aug. 2014.

[12] K. Chen, D. Shan, X. Luo, T. Zhang, Y. Yang, and F. Ren, "One rein to rule them all: A framework for datacenter-to-user congestion control," in *Proc. 4th Asia–Pacific Workshop Netw.*, Aug. 2020, pp. 44–51.

[13] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 632–647.

[14] Y. Xie, F. Yi, and K. Jamieson, "PBE-CC: Congestion control via endpoint-centric, physical-layer bandwidth measurements," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 451–464.

[15] M. Mathis and J. Mahdavi, "Deprecating the TCP macroscopic model," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 5, pp. 63–68, Nov. 2019.

[16] X. Qian, B. Wu, and T. Ye, "Two-level frame aggregation retransmission scheme design in 802.11n/ac/ad," *Xi'an Dianzi Keji Daxue Xuebao/J. Xidian Univ.*, vol. 45, no. 2, pp. 90–96, 2018.

[17] A. Masiukiewicz, "Throughput comparison between the new hew 802.11ax standard and 802.11n/ac standards in selected distance windows," *Int. J. Electron. Telecommun.*, vol. 65, no. 1, pp. 79–84, 2019.

[18] Y. Daldoul, D.-E. Meddour, and A. Ksentini, "IEEE 802.11n/AC data rates under power constraints," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[19] S. K. Saha, P. Deshpande, P. P. Inamdar, R. K. Sheshadri, and D. Koutsonikolas, "Power-throughput tradeoffs of 802.11n/AC in smartphones," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 100–108.

[20] J. Wu, B. Cheng, M. Wang, and J. Chen, "Quality-aware energy optimization in wireless video communication with multipath TCP," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2701–2718, Oct. 2017.

[21] S. M. ElRakabawy and C. Lindemann, "A practical adaptive pacing scheme for TCP in multihop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 19, no. 4, pp. 975–988, Aug. 2011.

[22] S. Zou, J. Huang, Y. Zhou, J. Wang, and T. He, "Flow-aware adaptive pacing to mitigate TCP incast in data center networks," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2119–2124.

[23] A. Pande and S. Devane, "Extensive simulation analysis of TCP variants for wireless communication," *Commun. Comput. Inf. Sci.*, vol. 969, pp. 529–542, 2019.

[24] K. A. Yadav and S. Kumar, "A review of congestion control mechanisms for wireless networks," in *Proc. 2nd Int. Conf. Commun. Electron. Syst. (ICCES)*, Oct. 2017, pp. 109–115.

[25] C. A. Grazia, N. Patriciello, M. Klapez, and M. Casoni, "A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control?" in *Proc. 14th Int. Conf. Telecommun. (ConTEL)*, Jun. 2017, pp. 75–82.

[26] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "A survey of congestion control mechanisms in Linux TCP," *Commun. Comput. Inf. Sci.*, vol. 279, pp. 28–42, Oct. 2014.

[27] M. Zhang *et al.*, "Will TCP Work in mmWave 5G cellular networks?" *IEEE Commun. Mag.*, vol. 57, no. 1, pp. 65–71, Jan. 2019.

[28] A. Parichehreh, S. Alfredsson, and A. Brunstrom, "Measurement analysis of TCP congestion control algorithms in LTE uplink," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2018, pp. 1–8.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] Y. Lin and V. W. S. Wong, "WSN01–1: Frame aggregation and optimal frame size adaptation for IEEE 802.11n WLANs," in *Proc. IEEE GLOBECOM*, Nov. 2006, pp. 1–6.

[35] 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-Business Telecommun.*, 2017, pp. 89–94.

[36] N. Cardwell. (Apr. 2018). *Linux TCP BBR Patch for Higher WiFi Throughput and Lower Queuing Delays*. RFC. [Online]. Available: https://groups.google.com/forum/#!topic/bbr-dev/8pgyOyUavvY

[37] N. Cardwell, "BBR V2: A model-based congestion control," in *Proc. ICCRG IETF Meeting*, p. 36, Mar. 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-%update-on-bbr-00

[38] C. Grazia, M. Klapez, and M. Casoni, "BBRp: Improving TCP BBR performance over WLAN," *IEEE Access*, vol. 8, pp. 344–354, 2020.

[39] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operat. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[40] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the internet," in *Proc. Int. Conf. Netw. Protocols*, 2000, pp. 177–186.

[41] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," Netw. Work. Group, Tech. Rep. RFC3782, 2004.

[42] (Jan. 2021). *Linux Kernel Patch, Source Scripts and Tests*. [Online]. Available: http://netlab.ing.unimo.it/sw/sourceton.patch

[43] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: Yet another highspeed TCP," in *Proc. PFLDnet (ISI)*, Feb. 2007, pp. 37–42.

[44] J. Corbet. (Aug. 2011). *Network Transmit Queue Limits*. LWN Article. [Online]. Available: https://lwn.net/Articles/454390/

[45] 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, 2019, pp. 19–22.

[46] C. A. Grazia, "A performance model for Wi-Fi frame aggregation considering throughput and latency," *IEEE Commun. Lett.*, vol. 24, no. 7, pp. 1577–1580, Jul. 2020.

[47] 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.

[48] H. Hassani, F. Gringoli, and D. J. Leith, "Quick and plenty: Achieving low delay and high rate in 802.11ac edge networks," 2018, *arXiv:1806.07761*.

[49] D. Taht and J. Gettys. (2014). *Best Practices for Benchmarking CoDel and FQ CoDel*. [Online]. Available: http://goo.gl/FpSW5z

[50] T. Høiland-Jørgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," in *Proc. 11th EAI Int. Conf. Perform. Eval. Methodologies Tools*, ValueTools 2017, Venice, Italy, Dec. 2017, pp. 1–6, doi: 10.1145/3150928.3150957.