Received July 21, 2021, accepted July 30, 2021, date of publication August 3, 2021, date of current version August 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3102022

Future of TCP on Wi-Fi 6

CARLO AUGUSTO GRAZIA, (Member, IEEE)
Department of Engineering Enzo Ferrari, University of Modena and Reggio Emilia, 41125 Modena, Italy
e-mail: carloaugusto.grazia@unimore.it

ABSTRACT The Linux TCP/IP stack contributions have recently pointed all in the same direction: maximize the available throughput while maintaining low latency. These activities started by mitigating the buffebloat phenomenon at the network bottleneck as much as possible. So far, the deployed solutions have been designed by considering standard models for the bottleneck that could be either wireless or wired. The introduction of Wi-Fi 6 is dislocating the bottleneck of standard home and office WLAN from the radio access point to the 1 Gbps interface, the wired interface that points to the internet service provider; this bottleneck migration leads to a new real-case bottleneck model, which is hybrid. Such an environment embraces new technologies and provides new challenges for the old TCP protocol when applied to hybrid bottlenecks and operating in conjunction with the TCP side-modules, which are now part of the novel Linux kernels. This paper aims to highlight the TCP performance considering the new TCP modules and novel scenarios opened by Wi-Fi 6 with real-case hybrid bottlenecks.

INDEX TERMS Congestion control, frame aggregation, latency, pacing, TCP, TSQ, WLAN.

I. INTRODUCTION

In the last ten years, several research activities in the Internet and networking field have focused on the ‘‘bufferbloat’’, which is a word to define the excessive buffering of packets causing high Internet latency. The mantra for mitigating this phenomenon is well defined and consists of maximizing the available throughput while maintaining latency as low as possible; to do so, two families of algorithms have been introduced, TCP related algorithms and queueing disciplines. The formers operate on the endpoints of a network path and include novel congestion controls like BBR (Bottleneck Bandwidth and Round-trip propagation time), and novel TCP modules like TCP Small Queues (TSQ) and TCP Pacing (TP). The latter can instead operate on every node of the path, operating at the queueing level, and can be divided into Active Queue Management (AQM) and packet scheduling. AQM algorithms have the precise goal of mitigating the formation of large buffers at the bottleneck, and they have been the first concrete solution implemented by the bufferbloat community to mitigate the end-to-end latency through the CoDel (Controlled Delay) algorithms family. On the other side, packet scheduling deals with QoS, prioritizing traffic, and helping to shape the bandwidth between competitive flows, which is a complementary task concerning this manuscript’s goal.

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Fu Cheng.

These novel algorithms have been modeled and tested on wired networks first, particularly in Google data centers. This is the case, for example, of BBR, which has incrementally replaced CuBic in the YouTube servers. Studying these algorithms on wired networks means to rely on standard bottleneck models. Indeed, there is a large literature study of these modules in wired networks; the reason is that they have been proposed, generally, for deployment on the server-side. Moving the focus to the user side, the configuration of the access network point changes due to the massive presence of Wi-Fi access technologies, which have a different bottleneck model with respect to the wired network. So far in FTTC/FTTH (Fiber To The Cabinet/Home) world, the scenario has been a WLAN deployed with IEEE 802.11n or IEEE 802.11ac technologies, with hundreds of megabits per seconds at the radio access, followed by a typical 1 Gbps interface to reach the Internet Service Provider (ISP). This scenario lacks tests, resulting in inefficiencies investigated in [1]–[3] to adapt current TCP solutions to deal with the Wi-Fi bottleneck. The critical difference between wired and Wi-Fi scenarios is the latter’s presence of a frame aggregation mechanism [4], [5]. Frame aggregation boosts the throughput by grouping packets to transmit in a single large frame, maximizing the radio efficiency but requiring some packets to be available at the interface. This technique introduces a trade-off between throughput and latency that must be considered when the Wi-Fi interface is the bottleneck one.

But what about now with the introduction of Wi-Fi 6 and IEEE 802.11ax technology? The main consequence of dealing with Wi-Fi 6 at the radio access is that we are moving the bottleneck again from wireless to wired. Due to the high radio capacity, the bottleneck is plainly the wired 1 Gbps interface, or better: the bottleneck can be modeled as a function of the frame aggregation of Wi-Fi 6, moving to the wired bottleneck model once a minimum threshold of frame aggregation is reached. We will refer to this as the hybrid bottleneck in this paper, which is depicted in Figure 2. The hybrid bottleneck needs a new model which better fits a real-case bottleneck and does not behave like a standard wired or wireless bottleneck. Moreover, so far, TCP congestion controls have been designed based on these standard wired or wireless bottlenecks, leaving an open challenge on the design of new TCP algorithms. On the one hand, sometimes it is better to enqueue more packets to create larger aggregates to increase the throughput despite the latency increment. On the other end, this mechanism must be avoided if a wired bottleneck is reached.

A. CONTRIBUTION

The contribution of this paper is to define and provide an analytical model of the hybrid bottleneck, introduced by Wi-Fi 6, as a function of the packet enqueued by the TCP congestion control and frame-aggregation size. Moreover, the model is validated through real tests on a novel Linux kernel involving state-of-the-art TCP congestion controls. The manuscript focuses on the endpoints of the communication path, where the TCP congestion control operates in conjunction with the TCP modules like TP and TSQ, which are fundamental to control the packets enqueued by the congestion control and understand the hybrid bottleneck characteristics. The final goal is to highlight the role of these algorithms, understanding how to control the TCP traffic performance over this new hybrid bottleneck.

In this manuscript, we will focus mainly on endpoints TCP solutions like TP and TSQ. Indeed, the contribution of this paper is to model the hybrid bottleneck introduced by Wi-Fi 6 and highlight the role of these algorithms in the performance of TCP traffic over this new bottleneck.

The rest of the paper is organized as follows: Section II describes the related work, while Section III details the possible network bottlenecks; in particular, Section IV models analytically the hybrid bottleneck of Wi-Fi 6. In Section V, the current TCP stack of Linux systems is described, and Section VI depicts the testbed used to produce the results analyzed in Section VII. Finally, Section VIII concludes the paper.

II. RELATED WORK

The hybrid bottleneck model comes from the standard models of wired and wireless bottlenecks defined in [6] and [7], respectively. This transition from wired to wireless analysis has been enabled by studies on the frame aggregation impact over the network performance, as provided in [4] where, in particular, the overhead of frame aggregation mechanism is investigated. Furthermore, an analytical model of a single IEEE 802.11n station in terms of expected throughput is presented in [5], as a function of the transmission time and the packet sizes.

Concerning the experimental results on Wi-Fi bottlenecks, the limit imposed by the TSQ in the wireless station upload has been investigated in [1], which has been the first literature contribution that captured this problem with TCP CUBIC. The reason for this limitation is indeed in the frame aggregation mechanism, which is disturbed by TSQ that limits the number of available packets in the station Network Interface Card (NIC). From the downlink side, Hassani *et al.* [8] elaborated on the server sending rate in IEEE 802.11ac networks, trying to maintain a steady aggregation size at the Wi-Fi bottleneck, controlling the throughput and latency tradeoff.

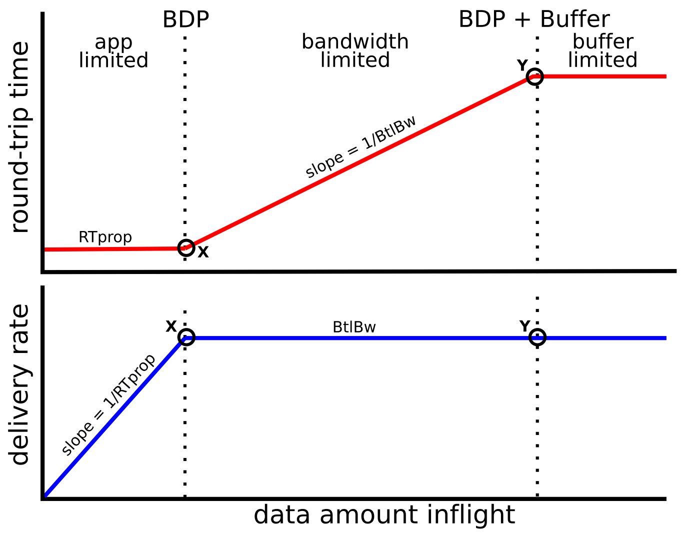
Additionally, the network congestion introduced by simultaneous uploads and downloads on a Wi-Fi network is considered in [9]; both long and short TCP flows and the Wi-Fi aggregation level are taken into account. The interest in network congestion and, in general, the real performance of TCP traffic is growing. To give an example, in [10] 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. At the same time, in [11], the authors analyzed the TCP's initial window in the wild, also in this case with a massive real test campaign.

Other research studies focus on the Linux kernel mainline driver for Wi-Fi environment, which is `ath9k`. In one work, the driver is modified to accommodate the FQ-CoDel solution in the firmware, controlling the Wi-Fi environment's bufferbloat phenomena without compromising the maximum aggregation size [3], while another work points on boosting the TCP fairness between competitive flows [2]. Finally, a TCP variant at the bases of BBRv2 [12], called BBRp [13], has been proposed to solve the BBRv1 inefficiency in Wi-Fi bottlenecks, allowing the congestion control to discover higher available bandwidth exploiting frame aggregation.

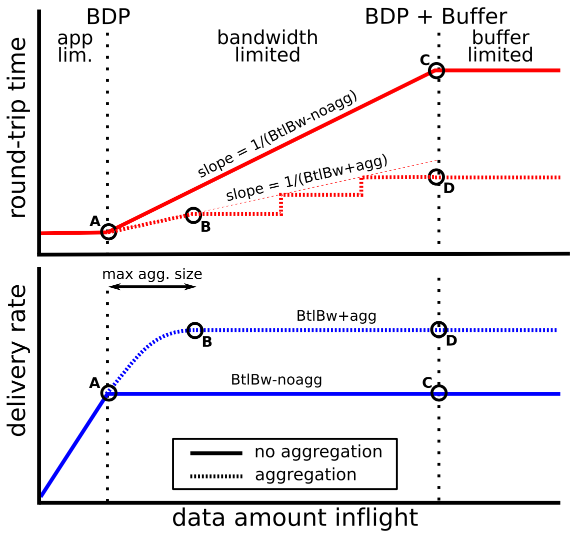
To the best of our knowledge, no literature contributions focus on different TCP variants together with TSQ or TP in Wi-Fi environments. This missing is crucial, in particular, with Wi-Fi 6, which impose a new role for the Wi-Fi interface, delegating back the bottleneck role to the standard 1 Gbps wired interface of the routers. This analysis is necessary because TSQ, TP, BBR, and FQ-CoDel (to name a few) have been introduced with a specific goal: mitigate the bufferbloat phenomena. Anyway, the interaction of these modules with new Wi-Fi modules in hybrid bottlenecks has not been investigated.

III. BOTTLENECKS IN A NUTSHELL

The model for the wired network bottleneck is known in the literature and has been used in concrete to design the



(a) Wired bottleneck.



(b) Wireless bottleneck.

FIGURE 1. Wired and wireless bottleneck.

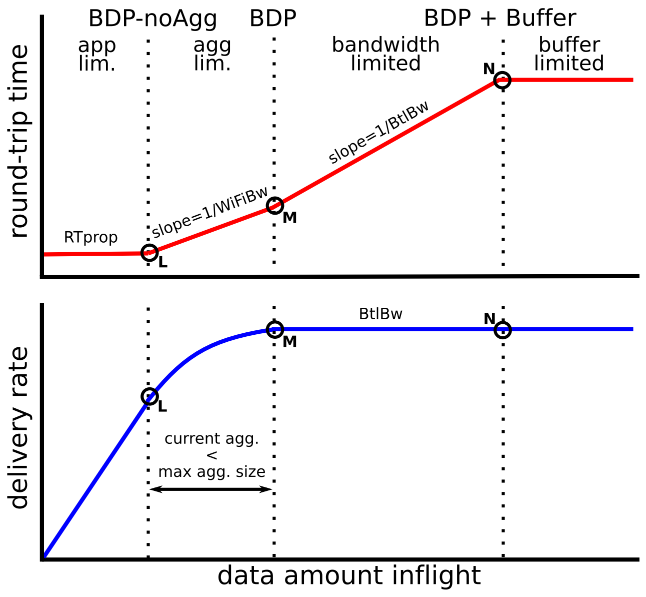


FIGURE 2. Hybrid bottleneck.

TCP BBR congestion control in [6]. This is not the same for a pure wireless bottleneck that has been recently modeled in [7] and reports the differences between a wireless interface with or without the frame aggregation mechanism implemented, which changes the behavior of the bottleneck in the central region macroscopically when the packets start to accumulate at the bottleneck queue. These two standard models are reported in Figure 1 with the wired bottleneck on the left (Figure 1a) and the wireless bottleneck on the right (Figure 1b). Figure 2, on the other side, defines the hybrid bottleneck in which the bottleneck moves from wireless to wired as a function of the aggregation.

A. WIRED AND WIRELESS BOTTLENECK

The classical wired bottleneck model reports the delivery rate (throughput) and the round-trip time (RTT) of a TCP connection as a function of the number of packets in flight, i.e., the number of packets delivered by the source and still waiting to be acknowledged. The throughput and the RTT are strongly coupled and bounded to the network behavior divided into three regions. In the application region, the bottleneck is software; this means that the connection can increase the throughput without affecting the RTT since the amount of data inflight can be absorbed by the network. The point `X`, known as Kleinrock's point, is the optimal operating point for a TCP connection and correspond to the amount of data inflight that matches the bandwidth-delay product (BDP) of the network. Transmitting more data than BDP has the sole effect of increasing the RTT since the packets start to accumulate at the bottleneck queue, while the throughput is capped at the maximum bottleneck speed named `BtlBw` in Figure 1a. Starting from the point `X`, the RTT grows linearly as a function of the amount of data inflight up to a point `Y`, where the maximum buffer size of the bottleneck is reached, and the packets start to be dropped. Once reached point `Y`, both the RTT and the throughput cannot grow anymore in ideal conditions. This model has been widely used for TCP congestion control designing; in fact, TCP BBR has the model goal to operate close to point `X` while standard loss-based congestion controls operate close to point `Y`, waiting for drops feedbacks to adjust the amount of data inflight.

The wireless bottleneck has the same behavior as the wired bottleneck if the frame aggregation mechanism is not deployed, and points `A` and `C` of Figure 1b are equivalent to points `X` and `Y` of Figure 1a. The implementation of the frame aggregation mechanism increases spectrum efficiency by sending more than one frame in a single transmission

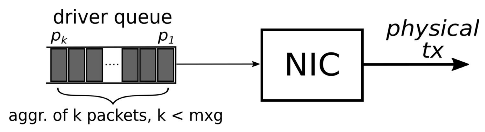


FIGURE 3. Aggregate of *k* packets in the driver queue.

opportunity, allowing the radio interface to increase the maximum throughput, i.e., increasing the `BtlBw`; additional details on the frame-aggregation mechanism can be found in [4], [5]. From point `A`, increasing the amount of data inflight and allowing frame aggregation increases both the throughput and the RTT. The idea is that transmitting more frames in a single transmission attempt allows increasing the throughput, non-linearly, by reducing the ratio between the payload and the radio access overhead. This benefit ends reaching point `B`, in which the maximum aggregation size is reached, and so also the maximum `BtlBw` in case of aggregation enabled. Starting from `B`, increasing the amount of data inflight has no impact on the throughput and can only increase the RTT introduced by the queueing delay in the wireless NIC queue, that accumulates packets. The packets are accumulated as a function of the amount of data inflight up the maximum buffer size of the wireless bottleneck interface in point `D`. After that point, the RTT cannot increase anymore, in ideal condition, and the packets start to be dropped. The next Section provides the curves' formula for throughput and RTT in points `A`, `B` and `D`.

B. HYBRID BOTTLENECK

Once defined the pure wired or wireless bottlenecks, it is possible to discuss a hybrid bottleneck where the throughput and RTT are affected by four different regions. Indeed, after the application-limited area, we have an aggregate limited region in which packets start to accumulate at the wireless NIC to create aggregates and increase the throughput, as well as the RTT as a side effect, moving from point `L` to point `M`. The connection can increase the throughput and exploit a higher `BtlBw` thanks to the frame aggregation, which takes advantages of the increasing number of data inflight to form larger aggregates and increase the throughput. This mechanism allows to exploit higher bandwidth up to point `M`, where the wired bottleneck limits the exploitation. This migration moves where the extra amount of packets inflight gets accumulated: it moves from the wireless interface (that allows at least a bandwidth equal to the wired interface with the current aggregation size) to the wired interface. Indeed, from point `M` to point `N` the throughput cannot increase anymore, limited by the wired bottleneck. The RTT increases as a function of the queueing delay at the wired bottleneck, in the same way as what happened between points `X` and `Y` of Figure 1a. It is essential to notice that this hybrid bottleneck can exist if and only if the maximum aggregation size is not reached, otherwise the bottleneck would be a pure wireless bottleneck as in Figure 1b.

TABLE 1. Aggregation parameters.

Notation	Description	Value
l	packet size	1500 bytes[2]
l_{oh}	physical overhead	48 bytes
r	station bit-rate: AX201NGW 2x2 MIMO	2.4 Gbps
r_{wired}	wired bottleneck bit-rate: Gigabit Ethernet	1 Gbps
mxg	max aggregation size	800 packets[3]
t_{oh}	channel access overhead	0.5 ms
RT_{prop}	base RTT propagation	2.5 ms

IV. HYBRID BOTTLENECK MODEL

In this section, we formalize the hybrid bottleneck model that describes analytically Figure 2. The model derives from the wireless bottleneck one reported in [7] that, instead, describes analytically Figure 1b. The main critical portion of Figure 2 is between the points `L` and `M`, which has the same non-linear curve of delivery rate of Figure 1b between points `A` and `B`. The absence of linearity, with respect to the previous known wired model of Figure 1a, is due to the presence of frame aggregation at the bottleneck NIC, which changes the delivery rate as a function of the bottleneck queue length, i.e., as a function of the aggregate size, the larger is the aggregate, the lower is the overhead and the higher is the throughput.

To model the frame aggregation mechanism and the hybrid bottleneck behavior, we refer to the parameters reported in Table 1. We assume the Wi-Fi 6 station to transmit at a constant bitrate r of 2.4 Gbps, which is the initial bottleneck of the hybrid system.[1] If the aggregate is composed of a single packet, the length of the frame transmitted will be:

$$\text{one packet frame size} = (l + l_{oh}) \cdot 8 \tag{1}$$

where $l \cdot 8$ is simply the conversion of a packet size from bytes to bits, and l_{oh} is the physical overhead introduced by Wi-Fi 6. If, instead, the number of packets at the bottleneck allows the formation of aggregate packets, like the situation depicted in Figure 3, the length of the frame transmitted will be:

$$\text{k packets frame size} = (k \cdot l + l_{oh}) \cdot 8 \tag{2}$$

where k is the number of packets included in the aggregate. The only constraint is that $1 \leq k \leq mxg$, where mxg is the maximum aggregation size imposed by the Wi-Fi medium.

Before continuing our analysis, it is important to clarify the hypothesis under which it is possible to talk about a hybrid bottleneck. The Wi-Fi delivery rate must exceed the wired bottleneck capacity with an aggregate of size $\bar{k}$ which is:

$$1 < \bar{k} < mxg.$$

In other words, $\bar{k}$ is the aggregation size at which the Wi-Fi throughput match the wired bandwidth bottleneck. This condition is crucial because if the wired bottleneck bandwidth is lower than the Wi-Fi bottleneck bandwidth without frame

[1]Without frame aggregation mechanism, indeed, even if the Wi-Fi 6 bitrate is 2.4 Gbps, the available TCP data-rate is less than 100 Mbps.

aggregation ($\bar{k} = 1$), it means we are in front of a standard wired bottleneck. In a similar scenario, the bottleneck model is simply the one depicted in Figure 1a, and the Wi-Fi hop is not introducing significant changes since it is never the bottleneck of the path. On the other side, if the wired bottleneck bandwidth is higher than the Wi-Fi bottleneck bandwidth at the maximum aggregation size ($\bar{k} = mxg$), it means that we are in front of a standard wireless bottleneck. In this scenario, instead, the bottleneck model is simply the one depicted in Figure 1b, and the wired path is not introducing significant changes since it is never the bottleneck of the path. If the initial condition holds, the Wi-Fi hop is the bottleneck until the aggregation size of k packets is below $\bar{k}$. In contrast, the wired interface becomes the bottleneck once the Wi-Fi reaches aggregation sizes of, at least, $\bar{k}$ packets. This bottleneck migration, from wireless to wired, introduces the need for a new model as a function of the aggregation size.

Now we need to formalize the delivery rate and the RTT of the hybrid bottleneck. To simplify the discussion, we refer to a steady-state case in which the NIC queue is backlogged. We start by describing the average transmission time needed by the Wi-Fi interface to transmit a generic aggregate of k packets:

$$TxTw(k) = \frac{(k \cdot l + l_{oh}) \cdot 8}{r} + t_{oh} \tag{3}$$

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 [4]. Similarly, the effective Wi-Fi throughput, assuming to work with only one station, without collisions and errors is:

$$Thr(k) = \frac{k \cdot l \cdot 8}{TxTw(k)} \tag{4}$$

which is simply the payload size divided by the time needed to transmit the associated frame. At the same time, the transmission time needed by the wired ethernet interface to transmit a generic queue of k packets is:

$$TxTe(k) = k \cdot \frac{(l + le_{oh}) \cdot 8}{r_{wired}} \tag{5}$$

where t_{oh} is not included since it is negligible on ethernet interface with respect to Wi-Fi ones, and le_{oh} is the host-to-network standard overhead for gigabit ethernet.

According to [7], the latency contribution at the Wi-Fi bottleneck is $2 \cdot TxTw(k)$, when $k < mxg$, so the RTT associated to the hybrid bottleneck can be easily formalized with:

$$RTT(k) = \begin{cases} RTT_{base} + 2 \cdot TxTw(k) & \text{if } k \leq \bar{k} \\ RTT_{base} + 2 \cdot TxTw(\bar{k}) + TxTe(k\text{-}\bar{k}) & \text{otherwise} \end{cases} \tag{6}$$

with the RTT that grows following the Wi-Fi slope, when the bottleneck is wireless ($k \leq \bar{k}$), and grows, instead, following the standard wired slope, once the bottleneck became the wired interface ($k > \bar{k}$).

Similarly, the hybrid bottleneck bandwidth is:

$$BW(k) = \begin{cases} Thr(k) & \text{if } k \leq \bar{k} \\ Thr(\bar{k}) & \text{otherwise} \end{cases} \tag{7}$$

since the available bandwidth grows with k, thanks to frame aggregation, up to $\bar{k}$ when the bottleneck migrates from the Wi-Fi to the wired interface without growing anymore.

V. LINUX TCP/IP STACK

The current and up to date TCP-IP Linux stack is depicted in Figure 4. We report the three main blocks involved in a TCP flow transmission with the whole TCP transport block on the left, the Queueing Layer that corresponds to the TCP-IP networking layer in the middle, and the host-to-network Driver block on the right. The role of TCP congestion control algorithm did not change recently, so the TCP socket is still calculating the congestion window (CWND) and dealing with the ACK reception according to the algorithm used. The most significant change in the last years has been in the way packets are delivered by the TCP socket, now regulated by TSQ and TP. These new submodules are reported in Figure 4. Once the TCP socket delivers the packets, they are enqueued in the lower layers. The Queueing Layer, depicted in the middle of Figure 4, deploys a standard FQ-CoDel algorithm, which is the default solution in recent kernels with many Linux distributions [14]. Once the scheduler delivers the packets, they move in the last block, where the driver firmware implements the last hardware queue before moving to the physical medium channel. Once a packet is physically transmitted, a completion signal is cross-passed to the TSQ algorithm.

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 *= tcp_pacing_ss_ratio;          // SlowStart phase
4: else
5:     rate *= tcp_pacing_ca_ratio;       // Cong.Avoid. phase
6: end if
```

A. TP AND TSQ

The most significant change experienced by the TCP-IP stack in the last years has been the introduction of TP and TSQ. The cooperative work of these two TCP submodules strongly impacts the way packets are delivered by the TCP socket, affecting the TCP RTT and the system latency. TP is controlled by two system variables, i.e., `tcp_pacing_ss_ratio` and

[2]We include in the packet size the MPDU delimiter size, MAC header, frame check sequence and padding for simplicity.

[3]According to the IEEE 802.11ax standard, the maximum size of an aggregate is 256 MPDU, which is more than 1000 ethernet frames. On Linux, this limit is further reduced to 4 ms of data at the current rate, that results in a limit of circa 800 packets with our `iwlwifi` driver.

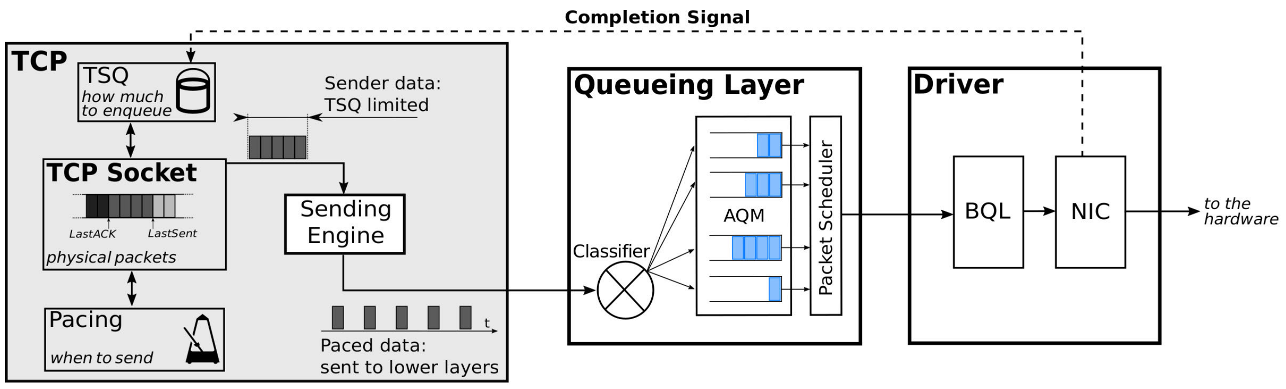


FIGURE 4. TCP-IP Linux Stack.

`tcp_pacing_ca_ratio`, used in the slow start and the congestion avoidance phases, respectively, as reported in Algorithm 1. The mathematical equivalence of Algorithm 1 is, instead:

$$\text{TCP\_paced\_rate} = \left(\frac{CWND \cdot MSS}{\text{base}_{RTT}}\right) \cdot pacing\_ratio \quad (8)$$

where MSS is the Maximum Segment Size and, consequently, $\frac{CWND \cdot MSS}{\text{base}_{RTT}}$ corresponds to the current TCP rate. The TCP socket's final TCP paced rate to deliver data is then adjusted with the `pacing_ratio` that changes according to the TCP transmission phase. By default, `tcp_pacing_ss_ratio` is equal to 2 in the slow-start phase, and `tcp_pacing_ca_ratio` is equal to 1.2 in the congestion avoidance phase. This means that the TCP flow doubles the slow-start phase rate and increases it by 20% in the congestion-avoidance phase. This mechanism allows probing for more bandwidth without forming excessive bursts of packets in the path's network queues.

```
Algorithm 2 TCP Small Queue
Input: TCP_SOCKET sk;
1: int limit;
2: limit = max(2 * sk→pktsize, sk→tcp_pacing_rate ≫ 10);
3: limit = min(limit, tcp_limit_output_bytes);
```

On the other side, the TCP paced rate is used to calculate, in conjunction with the TSQ mechanism, the number of packets that a TCP socket can enqueue in the sender stack. This quantity is a dynamic value that responds to the following equation:

$$\text{TSQ\_limit} = \min\Big(\texttt{bytes}, \max\big(\texttt{pkts} \cdot MSS, \texttt{ms} \cdot pacing\_ratio\big)\Big) \quad (9)$$

which is the mathematical equivalent of Algorithm 2, where `pkts` and `ms` are converted in bytes for consistency. Equation 9 guarantees that the TSQ limit (expressed in bytes) is always higher than a minimum amount of packets `pkts` (2 packets by default) and lower than a maximum amount of bytes `bytes` (128 KB by default). The dynamic limit moves through these two bounds and is the amount of data that corresponds to a latency equal to `ms`, 1 ms by default. Algorithm 2 clarifies this behavior: the dynamic amount of data that can be enqueued is calculated through $sk \rightarrow$tcp_pacing_rate $\gg$ 10, which is a 10-bit shift of the current pacing rate, that corresponds to the amount of data transmitted in 1 ms at the current paced rate. This mechanism helps the sender congestion control mitigate the queueing delay inside the node and accurately calculate RTTs. The bit shift quantity changes the latency introduced by TSQ, while the TP ratio changes the TSQ limit size.

B. QUEUEING LAYER AND DRIVER

The default structure of FQ-CoDel, reported in Figure 4, works as follows: a separate software queue serves each TCP flow, and each queue is managed by the CoDel algorithm to control the latency and is served in a round-robin fashion. The default CoDel threshold is set at 5 ms, which means that packets with sojourn time greater than the threshold will be dropped at the dequeue stage. The Queueing layer and the NIC Driver block are strongly coupled in their behavior. Indeed, through the usage of the `mq` Linux queueing discipline, one separate packet scheduler for each NIC hardware queue is implemented; Figure 4 represents a simple scenario in which a single hardware queue is present. The driver also implements the Byte Queue Limit (BQL) for all the hardware queues, which is the last algorithm to control the global latency of the system [15]. The BQL mechanism tries to store enough data to avoid starvation and, at the same time, tries to avoid accumulating excessive data increasing the latency. The BQL algorithm is not tested in our paper, and the drivers' default configurations are maintained, this means that the BQL is working during the experiments with default parameters.

VI. TESTBED

This section describes our testbed, which is depicted in Figure 5. Each test involves a single client, a server, and the WiFi 6 Access Point. The end nodes run the Arch Linux

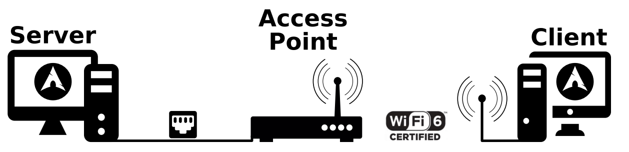


FIGURE 5. Physical testbed layout.

TABLE 2. Testbed parameters.

parameter	value
Kernel version	5.4-lts
Linux Distribution	Arch Linux
TCP Congestion Control	Cubic, BBR, BBRv2, New Vegas (NV), YeAH, New Reno, HighSpeed (HS)
TSQ type	`TSQ` (standard), `2TSQ`, `4TSQ` `8TSQ`, `16TSQ`, `32TSQ`
TP Rate	`1p` (standard), `2p`, `3p`
Queueing discipline	FQ_Codel
Wireless Chipset (router)	ASUS RT-AX92U 4x4 MIMO
Wireless Chipset (host)	AX201NGW 2x2 MIMO
Wireless Driver	`iwlwifi`
Tests	1-8 TCP Uploads, RRUL
Metrics	TCP Throughput, TCP RTT, ICMP Latency (`ping` RTT)

distribution with a 5.4-lts kernel version, while the access point is an ASUS RT-AX92U device with 4×4 MIMO that can provide Wi-Fi 6 connectivity with the IEEE 802.11ax standard. The testbed represents an FTTH environment that is widespread nowadays for both home and office networking with a high capacity Wi-Fi access network and a 1 Gbps wired interface through the ISP.

The client station connects to the access point through an AX201NGW chipset, with 2×2 MIMO, supported by the `iwlwifi` Linux driver. The client uses six possible TCP congestion control algorithms (reported in Table 2) and can set different possible TSQ limits 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. While standard TSQ allows each socket to enqueue "1 ms of data" at the current rate, our 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, and power of 2 integers are preferred. Moreover, static TSQ sizes can be imposed, which means that the actual TSQ value does not depend on the current rate but can be expressed in bytes or packets. The latter feature is used to verify the hybrid model described in Figure 2 by controlling at a fine-grained level the TSQ size and, consequently, the data amount in flight.

The other most critical parameter introduced and tested in this paper is the TP rate. TCP BBR does not react to any modification to the standard pacing value offered by the current Linux systems, so we used the BBRp patch [13] to allow BBR to use the same pacing rate of other congestion controls. The pacing rates used in this paper are named `1p`, `2p` and `3p`: where `1p` represents the standard pacing rate used by all the TCP variants, `2p` doubles the values and so on. Details about the BBRp patch and results not included here for space limitations, can be found in [16].

We followed the best practice document [17] 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/GRO/LFO). All of these adjustments serve to reduce sources of delay rather than those induced by the algorithms themselves.

All the experiments reported in this paper have been organized by using the Flent [18] tool, a flexible network tester that gives the possibility to manage different traffic typologies efficiently as well as to auto collect many performance metrics. Tests are organized as follows. We start a standard TCP flow in upload from the wireless client to the server. Each test runs for 40 seconds, of which five initial seconds with only ICMP traffic, 30 seconds for the actual TCP transmission, and five 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. Summarizing, all the parameters used to configure our experiments are summarized in Table 1 and Table 2.

VII. RESULTS

A. HYBRID MODEL: FROM THEORY TO PRACTICE

The first set of tests has been configured with static values of TSQ. This has been necessary to control the number of packets in flight at a fined-grain level, i.e., acting directly on the Wi-Fi bottleneck queue in the upload stream thanks to the TSQ property. By doing this, it has been possible to profile the x-axis of Figure 2 controlling it with TSQ values from 1 to 2048 packets with steps of 32. The TCP congestion controls used in this first test are TCP Cubic and TCP BBRv2 due to their different behaviors on pure bottlenecks. Each run has been repeated 20 times for collecting statistics. Figure 6 shows the result of this first experiment and reports both the TCP throughput and the latency. It is possible to notice how both the throughput and the latency of Figure 6 reflect the delivery rate and RTT curves of Figure 2; the points `M` and `N` of the model has also been reported on the collected data to simplify the reading. With a number of packets in flight close to 512, the Wi-Fi 6 station is able to reach the 1 Gbps bottleneck of the Ethernet wired bottleneck, indicating that 512 is $\bar{k}$ of our testbed, while mxg is circa 800 packets. This point is indicated with `M`, recalling the same point of the model depicted in Figure 2: from the point `M`, indeed, it is not possible to increase the overall throughput higher than 1 Gbps, due to the wired bottleneck, while the RTT can grow,

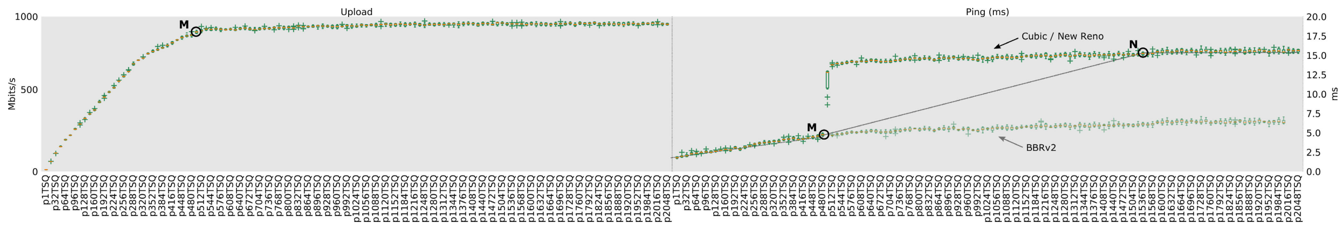


FIGURE 6. Throughput and RTT on Wi-Fi 6 hybrid bottleneck.

changing the slope, up to point N which is between 1405 and 1536 packets. The point N position reflects the hardware characteristics of our testbed, since the wired bottleneck is able to accommodate 1000 packets[4] and the Wi-Fi interface needs 512 packets to reach the wired bottleneck throughput, the number of packets inflight to discover point N must be the sum of the two. The real data of TCP Cubic and TCP BBRv2 does not follow the model from point M to point N, due to their intrinsic characteristics. TCP Cubic is a loss-based congestion control: once the bottleneck moves from wireless to wired, the control of the bottleneck queue length is not in charge anymore to the TSQ mechanism. This fact allows TCP Cubic to operate, as known from the literature, over a standard remote wired bottleneck, moving straight to the operating point N, reacting only in the presence of losses. TCP BBRv2, instead, is a model-based variant, and it is designed to maintain an operating point close to M, without filling the wired bottleneck queue, which does not provide extra throughput and only provides extra latency.[5] The only way to draw the model line from M to N, is controlling the drop limit at the wired queue, from 1 to 1000, and using TCP Cubic, which always operates filling the queue, but it is not the goal of this paper.

B. CHANGING TSQ SIZE

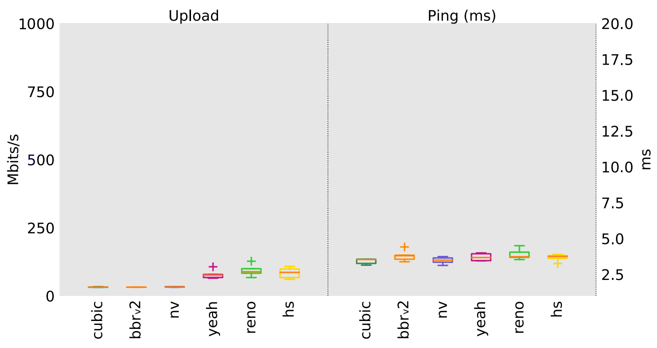
We continue our analysis using the standard representation of TSQ, where the TSQ size is expressed as a function of the milliseconds' parameter, starting first with tests in which the TSQ size is left at its default value of 1 ms. This first set of tests indicates that the initial bottleneck must be considered the Wi-Fi 6 interface, unable to exploit the available network bandwidth if frame aggregation is not properly utilized. We report the results of 6 selected TCP variants, namely Cubic, BBRv2, New Vegas, YeAH, New Reno, and HighSpeed, performing a TCP upload. 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 [16], not here included. Figure 7 shows that none of the different TCP variants is able to reach the available 1 Gbps of network bandwidth; all the congestion controls are indeed blocked close to 100 Mbps. The latency registered by all the TCP variants is very close to the base RTT of our network of 2.5 ms. The only difference is between the group formed by Cubic, BBR, and New Vegas that reports uploads slightly lower than YeAH, New Reno, and HighSpeed TCPs. This small gap resides in the TSQ limit interpretation of the different TCP variants, allowing the last three TCPs to aggregate a bit to take advantage of small aggregates and reach slightly higher throughput. These differences are reduced moving from a single TCP upload to 4 TCP uploads simultaneously active in the experiment reported in Figure 7b. This latter plot tells us two things: first, the intrinsic differences between the TCP variants disappear with negligible variations of throughput and latency between them, and second, increasing the number of TCP streams does not mitigate the limit of 100 Mbps imposed by TSQ due to the absence of effective frame aggregation at the Wi-Fi 6 NIC.

Our TSQ patch help to overcome the TCP upload throughput limit, relaxing the amount of ms for the TSQ and allowing almost all the different congestion controls to effectively aggregate packets at the Wi-Fi 6 NIC to exploit the available network data-rate. Figure 8 reports the same experiment of Figure 7, whit a single TCP upload, but including different TSQ sizes, from the standard 1 ms up to 32 ms. On one side, it is clear how relaxing the TSQ constraint helps many congestion controls reach the system's maximum data-rate; on the other side, it is clear how different TCP variants behave differently to this change.

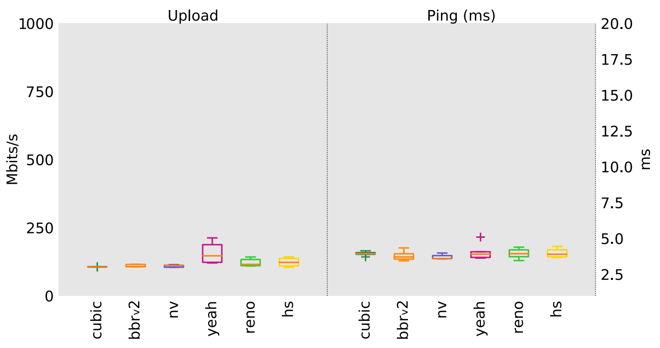
The first observation is on New Vegas, which through the TSQ relaxation, increases the throughput up to 350 Mbps without reaching the optimal value. The reason is the delay-based nature of TCP New Vegas, which cannot aggregate appropriately as the frame aggregation impacts the latency, increasing the queueing delay. Instead, almost all the other TCP variants can easily saturate the Wi-Fi 6 hop reaching the 1 Gbps limit imposed by the following wired hop through values of 16TSQ or 32TSQ. Some different behaviors are crucial by observing the latency; the loss-based variants Cubic, New Reno, and HighSpeed increase the latency between 15 and 17 ms with 16 and 32TSQ, due to their nature to operate at the point N of wired bottlenecks. The delay-based variant of New Vegas instead maintains the latency close to the base RTT. The only congestion controls that move in the trade-off region are TCP YeAH and TCP BBRv2: TCP YeAH reaches an almost optimal value of TCP throughput, increasing the latency more gently than

[4]For this experiment FQ-CoDel has been disabled at the wired bottleneck, to allow to appreciate the slope between M and N.

[5]We used BBRv2 instead of the original BBR to allow the congestion control to aggregate packets and reach the M point.



(a) 1 TCP Upload.



(b) 4 TCP Uploads.

FIGURE 7. TCP upload: Standard TSQ, Goodput vs. Ping.

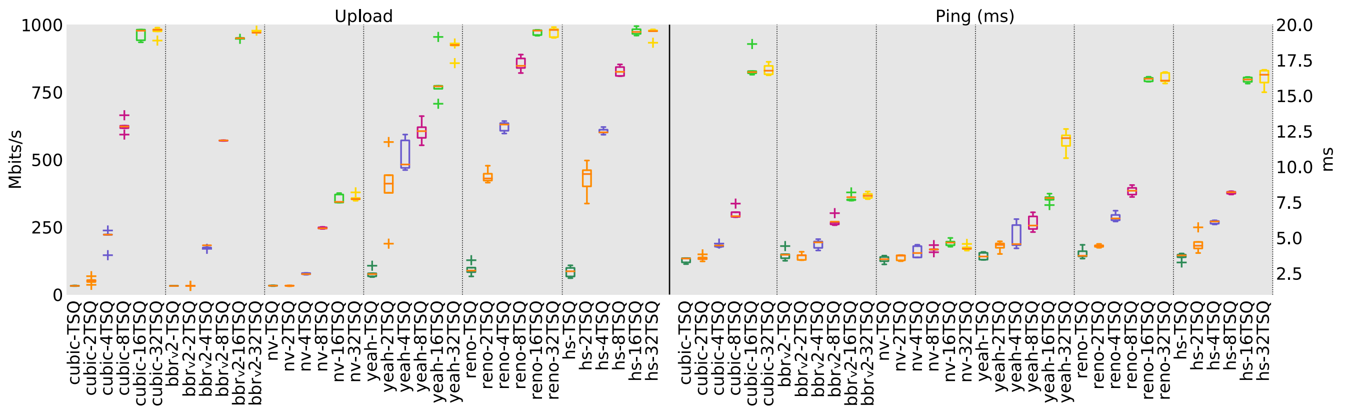


FIGURE 8. One TCP flow in upload: different TCP & TSQ, Goodput vs. Ping.

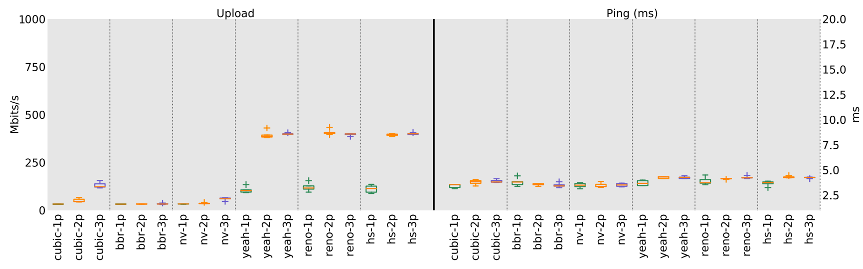
loss-based variants, while TCP BBRv2 manifests the best trade-off reaching the optimal throughput with 16 and 32TSQ like the loss-based TCP group, but maintaining a latency slightly close to 7 ms, which corresponds to the M operating point.

C. CHANGING TP

The second set of experiments aims to analyze the impact of TP on throughput and RTT. We considered again different TSQ sizes and different TCP congestion controls, replacing BBRv2 with BBR, since BBRv2 has a custom pacing engine, while with BBR we can appreciate different TCP pacing speeds as well as on the other congestion controls.[6] TP impacts the TSQ size computation, as seen in Section V, by boosting the ability of each TCP variants to discover higher data-rate available. Figure 9 shows the effect of increase the TP rate from `1p` to `2p` and `3p` on throughput and latency, including TCP RTT, maintaining a default TSQ configuration on a single TCP upload. The results are significant since TCP YeAH, TCP New Reno, and TCP HighSpeed reach almost 500 Mbps of throughput without significative impact on both `ping` latency and TCP RTT. Simultaneously, TCP Cubic reacts with a smaller effect to the TP changes increasing the throughput only partially. Even in this case, TCP BBR and TCP New Vegas privilege a low latency without effective frame aggregation.

It is now interesting to observe the same experiment reproduced with 4TSQ instead of the default one. The value of 4TSQ is interesting since it is the selected value from the wireless Atheros drivers in the Linux kernel mainline for allowing packet aggregation at the NIC. All the results involving other TSQ values are available in [16] and are not included here to avoid redundancy. In this case, all the TCP variants have a specific reaction to the TP change, thanks to the more relaxed TSQ value that leaves room for appreciating different behaviors. TCP Cubic manifests the largest hops in throughput from less than 300 Mbps with `1p`, up to a saturated value of 1 Bbps with `3p`, maintaining the latency and TCP RTT increment close to 4 ms. This helps to appreciate how much TCP Cubic is reactive to TP changes, and this is reasonable since the TP mechanism has been introduced when TCP Cubic was the default congestion control on Linux systems. TCP BBR increases the throughput relaxing the TSQ policy at 4TSQ, and combining it to a TP increment. The throughput reached is more than 600 Mbps with 4TSQ, without significative changes between `2p` and `3p`. TCP New Vegas has a good spectrum of results with 4TSQ, in which the

[6]Details on our second patch that allows BBR to react to global TP variables are reported in [16].


(a) Goodput vs Ping.

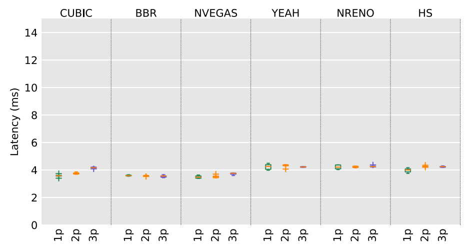
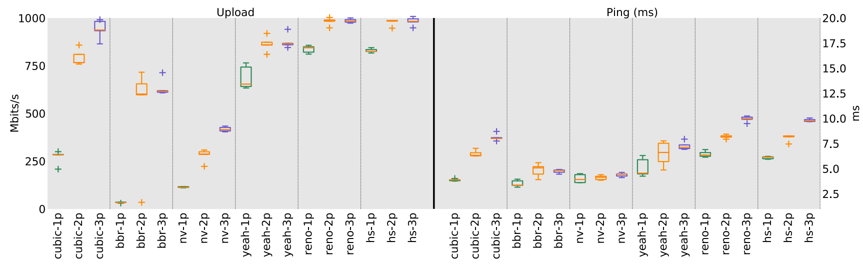

(b) TCP RTT.

FIGURE 9. Effect of TP: Standard TSQ.


(a) Goodput vs Ping.

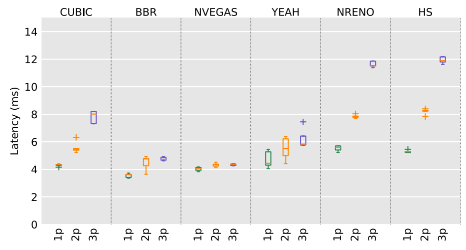

(b) TCP RTT.

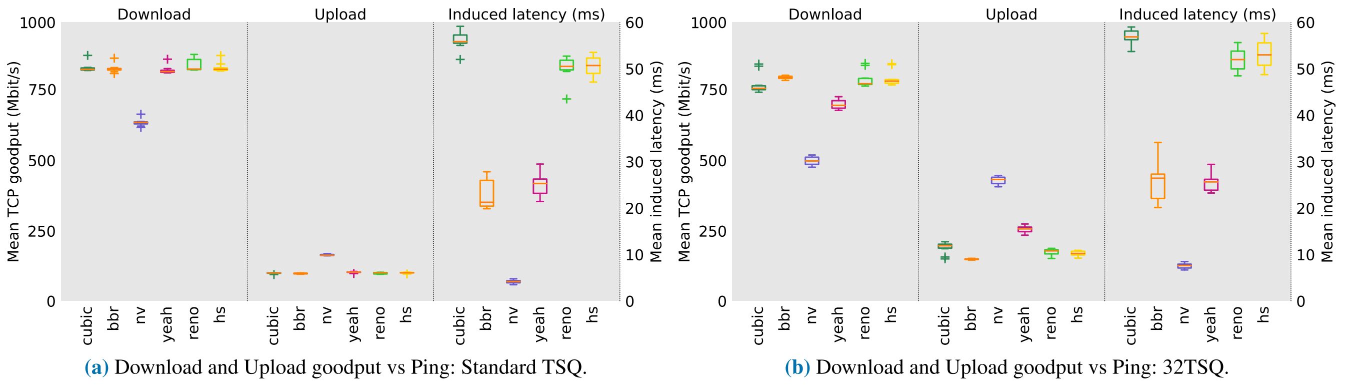
FIGURE 10. Effect of TP: 4TSQ.

difference between `2p` and `3p` can be appreciated moving to 250 and almost 500 Mbps of throughput, respectively. TCP Vegas provides the remarkable characteristic to avoid latency to become higher than 5 ms, which forbids the formation of the necessary aggregates to increase further the throughput. To conclude the description, again, TCP YeAH manifests the best tradeoff between throughput and latency with the 4TSQ configuration, reaching 850 Mbps with less than 2 ms of increment for both latency and TCP RTT. TCP New Reno and TPC HighSpeed manifest very similar behavior, moving from 800 Mbps with `1p`, to saturated values of circa 1 Gbps, with both `2p` and `3p`. With this experiment, it can be noticed that saturating the wired bottleneck using an aggressive configuration with high TSQ and TP leads only to a latency increment and even more TCP RTT, which, in this case, is greater than `ping` RTT. A similar conclusion is reached by TCP New Reno and TCP HighSpeed, indeed they start to saturate at 4TSQ with `2p`, which leads only to performance degradation in terms of TCP RTT that reaches 12 ms with `2p`.

D. REAL-TIME RESPONSE UNDER LOAD

We conclude our experimental results with the RRUL test, which is integrated into the Flent tool and consists of 8 streams of TCP traffic, 4 in download and 4 in upload, where we used the same TCP congestion control, changed each time as before. In other words, upload and download streams are always of the same TCP variant in each test, avoiding to incur in friendliness problem between a specific couple of TCP variants. Together with the 8 TCP streams, the RRUL test also creates an ICMP and a UDP traffic simultaneously. The Bufferbloat community has indeed proposed the RRUL test to challenge a network with heavy traffic load, leading to buffers' formation at the bottleneck and packet drops, enabling the evaluation of a network, and internet protocols under realistic congestion scenarios.

Figure 11 shows the performance of our network under the RRUL test, reporting the average throughput of the four upload streams, the four download streams, and the average `ping` RTT. We replicated the experiment with standard TSQ configuration and 32TSQ. Change the TSQ limit at the server has no impact on the results since the server is attached to the network through a wired ethernet connection and has no limitations related to TSQ and frame-aggregation mechanisms. The first result, in Figure 11a, manifests a huge unbalance between download and upload, with a distribution of data-rate which is close to 850 Mbps in download and 150 Mbps in upload; this happens for two reasons, the difference in terms of hardware characteristics between the access point and the client station, that privileges the download stream, and the presence of TSQ at the client, that mitigate the upload capacity according to what we have seen in the previous sets experiments. Simultaneously, despite the presence of standard TSQ, the `ping` latency is very high due to the congestion at the bottleneck wired link of the access point, leading to values of 50 ms for loss-based congestion controls like TCP Cubic, TCP New Reno, and TCP HighSpeed. At the same time, TCP BBR and TCP YeAH contain the latency to values close to 20 ms. The only exception is TCP New Vegas, which mitigates the latency to 5 ms and shares slightly better the available data-rate between the downstream and the upstream. The second result, in Figure 11b, reports the RRUL



(a) Download and Upload goodput vs Ping: Standard TSQ. (b) Download and Upload goodput vs Ping: 32TSQ.

FIGURE 11. **RRUL test: Standard TSQ vs. 32TSQ.**

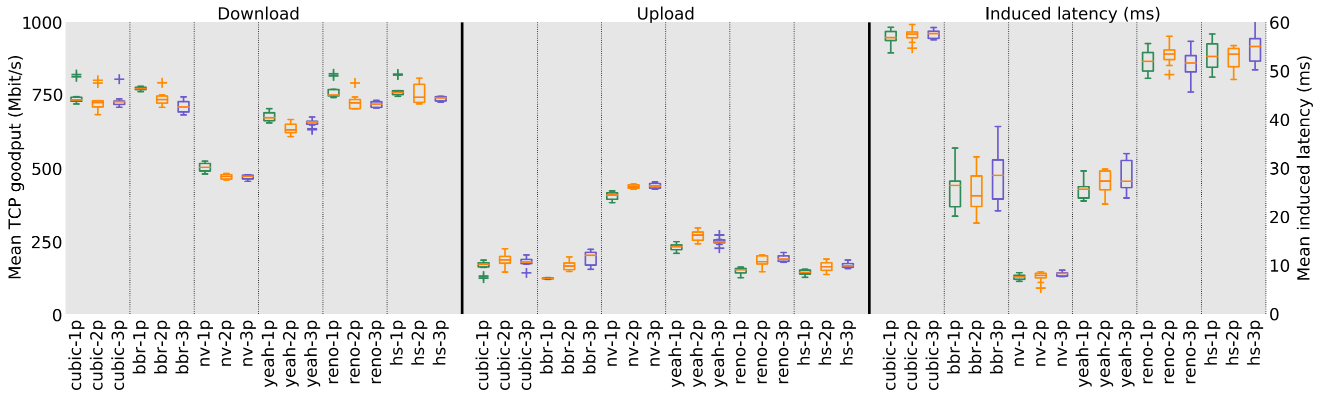


FIGURE 12. **RRUL test with 32TSQ: varying TP.**

test when the TSQ at the client station is relaxed at 32TSQ. The overall latency is not impacted, with negligible changes moving from standard TSQ to 32TSQ. Indeed, the larger contribution of `ping` RTT comes from the congestion at the wired bottleneck. From a balancing point of view between downstream and upstream, the impact of TSQ is evident, but still not enough to mitigates the hardware differences between the access point and the stations. Anyway, the gap between download and upload is reduced from standard TSQ to 32TSQ, for all the TCP variants. The only TCP variant able to split, almost equally, the data-rate in the two directions is TCP Vegas. The reason resides in the non-aggressive nature of the TCP congestion control, which cares only about the latency, leaving room for the upstream to grow and split the bandwidth with the downstream without taking advantage of the hardware characteristic.

To conclude the analysis, we also observed the impact of TP on the RRUL test with 32TSQ. The results are reported in Figure 12. Despite the effect of increasing the TP has a minor impact in this experiment with respect to the previous ones, it is mainly visible in TCP BBR, which a significant throughput increment in the upload stream, followed by a correlated downstream throughput reduction. All the congestion controls manifest the same effect, but with a minor impact with respect to TCP BBR.

VIII. CONCLUSION

In this paper, we have presented, analyzed, and tested the TCP performance on Wi-Fi 6, including the most up-to-date modules of the TCP-IP Linux kernel, such as novel and different TCP congestion controls, TCP Small Queues, and TP. Thanks to the characteristics of Wi-Fi 6 and current Internet networks, we defined and modeled the concept of a hybrid bottleneck, where the Wi-Fi interface is initially the wireless bottleneck, up to the formation of a certain level of frame aggregation that allows the Wi-Fi 6 to reach the typical 1 Gbps of wired Ethernet bottleneck. The bottleneck migration has been modeled analytically and proved by experimental results. Continuing, the paper analyzed the performance of different TCP over this hybrid bottleneck, investigating it as a function of the TSQ size, a fundamental parameter to allow the wireless interface to match the wired bottleneck, and TP. Results show that relaxing the TSQ standard limit is fundamental to exploit the available Wi-Fi 6 throughput with TCP uploads. The optimal balance between TSQ relaxation and higher TP depends on the TCP variant used, with TCP YeAH and TCP BBRv2 that interoperate well with the tradeoff between throughput and latency increment. The first scenario considered, with only TCP uploads in place, has highlighted the performance of TCP YeAH. The reason is the nature of TCP YeAH, a hybrid algorithm which interoperates well with the tradeoff between throughput and latency increment. In the second scenario considered, with heavy congestion through the RRUL test, the sole TCP able to guarantee fairness between downstream and upstream has been TCP New Vegas, which also manifested the lowest latency, close to the base network RTT, despite the high congestion. This paper poses the basis for optimizing TCP performance in novel Wi-Fi 6 environments and future ones, highlighting the role of fundamental end-node algorithms to control the latency and exploit the throughput, finding the optimal tradeoff as a function of the frame aggregation.

REFERENCES

[1] 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.

[2] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom, "The good, the bad and the WiFi: Modern AQMs in a residential setting," *Comput. Netw.*, vol. 89, pp. 90–106, Oct. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615002479

[3] T. Hoiland-Jorgensen, M. Kazior, D. Taht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," in *Proc. USENIX ATC*, 2017, pp. 139–151.

[4] 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.

[5] M. Kim, E.-C. Park, and C.-H. Choi, "Adaptive two-level frame aggregation for fairness and efficiency in IEEE 802.11n wireless LANs," *Mobile Inf. Syst.*, vol. 2015, pp. 1–14, Jan. 2015.

[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] 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.

[8] 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*. [Online]. Available: http://arxiv.org/abs/1806.07761

[9] S. R. Pokhrel, H. L. Vu, and A. L. Cricenti, "Adaptive admission control for IoT applications in home WiFi networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2731–2742, Dec. 2020.

[10] J. Zhou, Z. Li, Q. Wu, P. Steenkiste, S. Uhlig, J. Li, and G. Xie, "TCP stalls at the server side: Measurement and mitigation," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 272–287, Feb. 2019.

[11] J. Ruth, I. Kunze, and O. Hohlfeld, "TCP's initial window—Deployment in the wild and its impact on performance," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 2, pp. 389–402, Jun. 2019.

[12] N. Cardwell, "BBR v2: A model-based congestion control," in *Proc. ICCRG IETF 104th Meeting*, Mar. 2019, pp. 1–36. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-%update-on-bbr-00

[13] C. A. Grazia, M. Klapez, and M. Casoni, "BBRp: Improving TCP BBR performance over WLAN," *IEEE Access*, vol. 8, pp. 43344–43354, 2020.

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

[15] 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.

[16] (Jun. 2021). *Linux KErnel Patches, Source Scripts and Tests*. [Online]. Available: http://netlab.unimore.it/sw/sourceHB.zip

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

[18] T. Hoiland-Jorgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," in *Proc. ValueTools*, 2017, pp. 120–125.

CARLO AUGUSTO GRAZIA (Member, IEEE) 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 the 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, queuing algorithms, and V2X.