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

Ranganath B M[1], Ranjana Prasad C H[2], Rishi V N[3], S M Rachana[4], Mr. Pradeep Nayak[5]

[1,2,3,4]Student, Department Of Information Science And Engineering, Alva's Institute Of Engineering And Technology, India.

[5]Guide, Department Of Information Science And Engineering, Alva's Institute Of Engineering And Technology , India.

ABSTRACT

A number of Linux kernel modules have been added to the TCP/IP stack since the buffer bloat phenomena was defined, although there aren't many experimental research on their impacts.
when used with WLAN technology, specifically IEEE 802.11n and IEEE 802.11ac. TCP Small Queues (TSQ) is a crucial method that limits how many packets a TCP socket can queue in the stack. It does this by waiting for the physical layer to transfer the packets before enqueuing additional data. TCP Pacing (TP), a second important TCP mechanism, controls the speed at which the socket enqueues packets in the stack, regulating the formation of bursts of data. These methods impair throughput and have an impact on WLAN networks' frame aggregation logic.
trade-off between latency and all TCP variations. The performance of several TCP congestion control versions on wireless networks in the presence of various TSQ and TP policies is investigated experimentally in this research, which also models their interaction.

INTRODUCTION

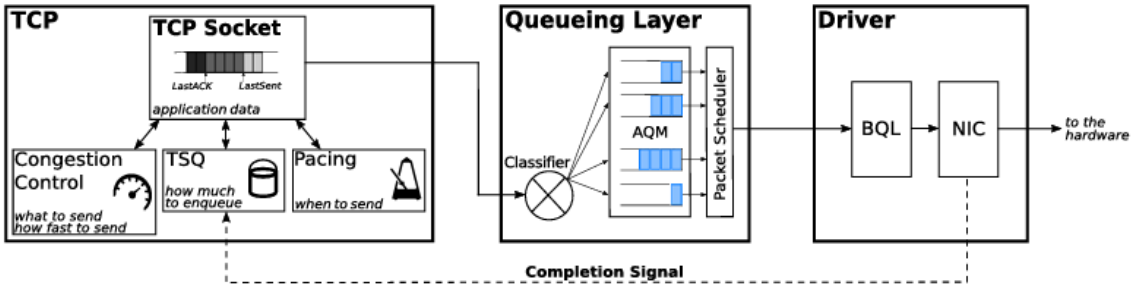
The growing use of Wi-Fi promotes both the creation of new, optimized standards and the improvement of existing ones [1] – [3]. As an example, IEEE Frame aggregation is required for 802.11n/ac/ax to increase overall throughput. By delivering many frames at a single transmission opportunity, frame aggregation improves spectrum efficiency [4]. Concurrently, the inflation of buffers along network channels meant to reduce link failure rates resulted in the now-well-known buffer bloat issue [5], which could result in unnecessary queuing delays of up to seconds. Since the Linux kernel is currently used by the great majority of servers connected to the Internet due to its speed, security, and robustness, we use it as the networking reference point in this study. Numerous options covering the whole Linux networking in order to improve network performance, stacks have just been developed. Proposals for the TCP congestion management domain at the transport layer include Google BBR [6], TCP Pacing (TP) and Small Queues (TSQ). The terms "bottleneck bandwidth" and "round-trip propagation time" are the sources of the acronym BBR. In fact, the BBR TCP congestion control algorithm has been developed to effectively utilize the network channel while controlling latency; BBR accomplishes this by creating a model of the network path in order to prevent and react to actual congestion. Every TCP congestion control method includes TSQ, a procedure intended to restrict the quantity of packets that can be queued down the stack at any given moment based on how quickly frames are efficiently sent by the Network Interface Card (NIC). The third module, TP, regulates the enqueue's speed. Together, TSQ and TP control the amount of data to be queued and the speed at which it is queued. Instead, a hybrid packet scheduler and Active Queue Management (AQM) algorithm dubbed FQ-Co Del [7] has been implemented at the network layer to directly solve the buffer bloat issue. By selectively discarding packets that stay in the queue for an extended period of time, FQ-Co Del stops the creation of big buffers.Although TSQ and TP function exceptionally well over wired lines, they may have an impact on the frame aggregation logic of wireless standards that employ them. In [8], we examined this restriction To the best of our knowledge, a precise investigation of the effects of TSQ and TP on TCP congestion controls like BBR and other TCP variations present in the Linux kernel is currently lacking in the literature when it comes to Cubic combined with TSQ. This study examines these effects on IEEE 802.11n and IEEE 802.11ac, two wireless technologies, both separately and in combination. For these standards to properly utilize the spectrum, frame aggregation is required. They are currently the most popular among those that need frame aggregation. Our tests demonstrate that the only effective strategy to use Wi-Fi bandwidth for uploads is to loosen the TSQ limit and TP, enabling increased throughput and the development of frame-

aggregation. The method is not a panacea and necessitates fine-grained customization based on the TCP variant and aggregation size employed by the particular technology being studied, which impacts the throughput-latency tradeoff. For the IEEE 802.11n and IEEE 802.11ac technologies, we chose two particular TSQ values that are now part of the Linux kernel mainline for the Atheros drivers. IEEE 802.11ac and 02.11n technologies. The remainder of the document is structured as follows: The relevant work is described in Section II, and the current Linux TCP/IP stack is shown in Section III. The interaction is modeled in Section IV. using the frame aggregation technique, TP, and TSQ. The testbed used to generate the results examined in Section VI is described in Section V. The paper is finally concluded in Section VII.

RELATED WORK:

Performance is constantly monitored by TCP congestion controls. The contributions to the literature include a wide range of scenarios with various topologies and technology. causing a number of problems. For instance, in [9], Zhou et al. categorized and examined server-side pauses pertaining to a basic TCP feature: the Retransmission Time Out (RTO) number. In the real world, RTO can have an impact on TCP performance. By examining multiple real-traffic TCP traces, the authors discovered this problem and came to the conclusion that customized RTOs were required for various services. The authors of another recent work [10] suggested a novel approach for scalably evaluating TCP congestion controls, which led to to a list of bugs that have never been reported. Finding the TCP variant operating on a network presents another difficulty. For example, the authors of [11] offered a method to determine the TCP congestion control employed by a distant server.The literature has extensively examined how TCP congestion affects performance. While some contributions concentrate on many niches, many cover the essential elements. A few original contributions are designed to train the congestion control algorithm across a wide range of networks using machine learning. This is the case for, to name a few, [12]–[14], which are pushing the research on this subject by contrasting the performance with current congestion controls. Contributions that completely rethink the TCP protocol and deprecate its macroscopic model with the advent of BBR are also included [15]. To the best of our knowledge, however, a pertinent component is absent. This is a thorough examination of the TSQ and TP algorithms as well as an experimental analysis of new TCP variations across WLANs. TP, TSQ, and other innovative algorithms such as FQ-CoDel and BBR, have been implemented to lessen the bufferbloat phenomenon. Nevertheless, there are unstudied disadvantages to using modern Wi-Fi modules.Using a bottom-up strategy, we now provide pertinent studies that are comparable to our contribution. The WLAN frame-aggregation logic has been thoroughly examined. supplied by simulations in [16]. Additionally, [17] provides a performance comparison using the ns-3 Network Simulator between the upcoming IEEE 802.11ax and the existing IEEE 802.11n/ac technologies. Another simulation-based study [18] examines the IEEE 802.11n/ac data rate under power limitations. These are all instances of related works that are unable to look into potential problems resulting from TSQ and TP because of the lack of of these mechanisms into the simulation environment of ns-3.However, there are actual TCP over Wi-Fi testing. An experimental assessment of the power-throughput tradeoff of IEEE 802.11n/ac technologies, including smartphones in the testbed, is presented in [19]. Experimental results even support articles pertaining to Multi-Path TCP (MPTCP) [20]. Nevertheless, the only information that is currently accessible indicates that TSQ violates IEEE 802.11n/ac's frame-aggregation logic. is [8], which excludes TP analysis and solely offers Cubic-related statistics. Ten years of research have been done on the mechanism of TCP pacing. This is the case of [21], where TP is examined in multi-hop ad hoc wireless networks, and [22], where the same mechanism is examined for data center network solutions.Moving on to broad assessments of several TCP variations over wireless networks that do not address the TSQ/TP algorithms and the frame-aggregation problem, there are primarily simulation-based works [23], [24]. Conversely, it is simple to locate works based on both simulations and actual testbeds for wired systems [25], [26]. This group includes even the learning-based congestion controllers. Although the implementation of PBE-CC [14] is a proof-of-concept that is not available in the Linux kernel for real-test comparison including TSQ and TP modules, it was created for cellular networks, where the environment is quite dynamic. However, the implementation is a proof-of-concept that isn't available in the Linux kernel for TSQ and TP module real-test comparison. The same debate for [13], which is predicated on surroundings that are imitated. One potential learning-based congestion control method, Rein [12], was even developed using an outdated kernel version 4.14 that was never added to the mainline. Several scientific studies have examined BBR performance using various methods. For instance, some recent research has attempted to address the question, "Will TCP work in mm Wave 5G cellular networks?" One study [27] is simulation-based and incorporates BBR between the TCP congestion controls under investigation. The difficulty of utilizing the available bandwidth during

"irregular" time intervals for the currently available TCP variants is a general problem for mm Wave.Keeping with the subject of cellular networks, BBR has been tested in [28] and [29] using the current 4G available technology; both the According to research, BBR performs better than New Reno and Cubic in terms of throughput and latency, but it has trouble preserving fairness amongst f lows under certain network conditions. Regarding fairness, this work [30] demonstrates that it is difficult to ensure justice between BBR and Cubic when various RTT flows are in place, but the unfairness gap can be significantly decreased thanks to FQ-Codel [7]. Modest-BBR [31] is a kind of BBR that increases fairness with Cubic and decreases aggressiveness while retaining comparable performance to the original BBR. To sum up, the next two studies examine the behavior of mixed BBR and Cubic traffic by addressing BBR's internal parameters, specifically its cycle [32]. In [33], BBR and Cubic have been tested over normal Gigabit Ethernet wired networks with a 4.9 kernel version, taking into account actual tests of BBR over Linux systems. The study demonstrates that when numerous flows are present, BBR does not exhibit its typical behavior in terms of fairness and latency reduction because of high queue occupancy. Conversely, frame aggregation across WLAN technology has primarily been studied using analytical models and simulations, first on IEEE 802.11n in [34] and more recently on IEEE 802.11ac in [35].



Moving on to the enhancement of BBR performance, a number of scientific studies attempted to address well-known BBR problems pertaining to Wi-Fi inefficiencies and RTT fairness. Google itself suggested a patch for the latter. first via the BBR-DEV RFC [36], and subsequently BBR v2 is a new version of BBR that has been accessible since Linux kernel version 5.x [37]. When the receiver is on a Wi-Fi network and the TCP sender is connected via Ethernet, BBR-DEV can function to increase the BBR throughput. In order to keep the bottleneck in use, BBR-DEV tells the sender to deliver additional data. Additionally, in order to reduce queuing delays, BBR-DEV introduces an adaptive drain approach. Instead, BBR v2 seeks to leverage a customized TCP pace to boost frame aggregation and enable BBR to advance throughout the WLAN situations, irrespective of wired or wireless sender connectivity. Additionally, we suggested a BBR variation called BBRp [38], which modifies the BBR cycle to address the pacing gain and draining phase. In order to improve performance in WLAN situations without sacrificing the BBR model-based nature in wired ones, BBRp aims to enable frame aggregation without unduly expanding the bottleneck queue. Tests where the transmitting node is not directly connected to a Wi-Fi interface and the Wi-Fi hop is just on the download are included in the BBRp paper [38]. way; this obscures the effect of TSQ, and nearly every TCP method exhibits comparable outcomes, with the exception of BBR v1, which is unable to aggregate correctly because of pacing problems.In summary, TP is essential to the community's efforts to enhance BBR's performance over WLANs. Conversely, there are no projects. that demonstrate the interference between various TCP variations and TSQ or TP. In order to bridge the gap, this book focuses on the difficult WLAN case that requires a frame aggregation mechanism.

LINUX TCP/IP STACK:

The Linux kernel's current TCP/IP stack is described in this section, along with all the additional components discussed in this paper, such as TSQ and TP, the Queueing Layer (QDisc), and the
Figure 1 shows all of the driver blocks. Three algorithms make up the present Linux TCP module: TCP Congestion Control, TCP Small Queues, and TCP pacing. The TCP Socket, which creates the TCP segments and controls the ACKs, sits on top of this module. Each TCP connection is associated with a particular TCP socket, and the three techniques are used to manage the packets.

A. Management of Congestion

It is a well-known component of the TCP module with a wealth of literature contributions regarding potential methods that can be utilized to manage traffic. The current Linux default, Cubic [39], BBR [6] (and three of its variants), created by Google and gradually implemented in numerous nodes, New Vegas [40], a timestamp-based congestion control similar to BBR, two window-based variants, New Reno [41] and HighSpeed (other TCP variants, such as Westwood+, Reno, Bic, Illinois, and Scalable, available in [42], are not included here due to space constraints), and a window-rate hybrid TCP variant, YeAH [43]. The methods used by these algorithms differ greatly: New Vegas and BBR are rate-based variations where the idea of time is focused on minimizing latency, whereas Cubic and the others are primarily window-based and prioritize maximizing goodput. Basic functions including calculating the transmission rate and congestion window size (CWND), as well as calculating the TCP parameters in the event of congestion events or packet loss, are handled by each congestion control.

B. TSQ, or TCP Small Queues

Google developed this method to decrease TCP flow latency. Each TCP socket can only queue a certain number of packets in its node stack thanks to the TSQ algorithm. By preventing the buildup of packets in the sender node queues, the objective is to lessen the Bufferbloat [5] phenomenon. The TCP socket is notified and permitted to enqueue a new packet on the stack only after the NIC completes the packet's dispatch. Each TCP socket can enqueue a certain amount of packets using the standard TSQ technique, which is equal to the number of packets corresponding to one millisecond of latency at the current sending rate; as a function of the f low throughput, this method sets an upper bound on the sending node queueing delay. It has been demonstrated in [8] that this worldwide constraint of 1 ms is excessively stringent in a Wi-Fi setting, where frame aggregation is inefficient under such a limit.

C. Pacing using TCP (TP)

The speed at which packets are pushed from the TCP module to the lower layers of the stack is specified by the algorithm. WhileTP restricts the internal rate at which packets are transferred to the networking layer, TSQ restricts the quantity of packets queued,

Algorithm 1 TCP Pacing Rate.

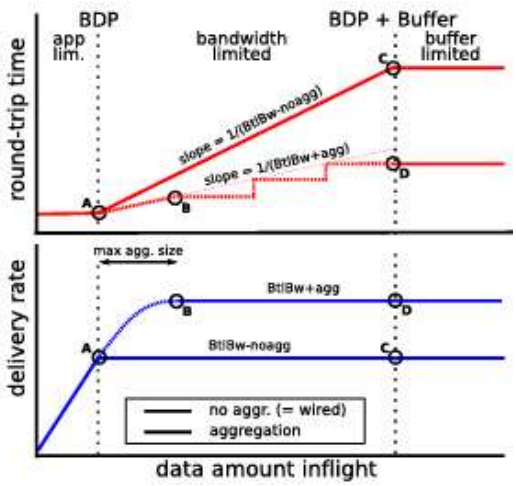
Input: TCP_SOCKET sk, **int** $base_{RTT}$;

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

Algorithm 2 TCP Small Queue.

Input: TCP_SOCKET sk;

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



imposing a time delay between enqueued packets. The Bufferbloat effect can be lessened by using both TSQ and TP to prevent burst generation. The goal of TP is to disseminate packets in the time domain using the congestion control's base RTT. In fact, the network's base RTT is a crucial component of the pacing algorithm. With the exception of BBR, which implements its pacing method, nearly all TCP congestion controllers employ a basic TP algorithm. The two default pacing rates used by the standard Linux TP algorithm are both represented as a percentage of the TCP flow's current rate: 120% and 200%. The former allows packets to be queued at a rate twice that of the current one during the slow-start phase, whereas the latter is employed to enqueue packets at a pace 20% faster than the existing one during the congestion avoidance phase. BBR employs a comparable value called TP Gain, which is 25% greater than the present rate and is hardcoded in the BBR algorithm and cannot be adjusted in userspace.

D. Interaction between TSQ and TP

These two TCP submodules' collaborative efforts have a significant impact on how the TCP delivers packets socket. The two TP variables that were previously discussed are tp_ss_ratio and tp_ca_ratio, which are employed in the congestion avoidance and slow start phases, respectively. Algorithm 1 illustrates how the TCP socket's final TCP timed rate to send data is subsequently modified using a tp_ratio that varies based on the TCP transmission phase.

1. After converting the previously specified percentages, tp_ss_ratio equals 2 and tp_ca_ratio equals 1.2, respectively, allowing to search for additional bandwidth without creating large packet bursts in the network queues.Conversely, TP and TSQ specify how many packets a TCP socket can queue in the sender stack.This value is dynamic and is determined using By default, Algorithm 2 uses the quantity of data that corresponds to a latency of one millisecond. Additionally, TSQ is limited by a minimum of two packets (by default) and a maximum of 128 KB of bytes. This behavior is explained by Algorithm 2, which computes the dynamic amount of data that can be queued using tp_rate 10, a 10-bit shift of the present pace rate, which is equivalent to a lag of one millisecond. This method lowers the RTT by assisting the sender congestion control in mitigating the queueing delay within the node.The typical structure of the FQ-Codel [7] algorithm, which is the default setting in many Linux versions, is likewise shown in Figure 1. After a packet is formed by the TCP socket, the packet enters the QDisc layer, where AQM policies and packet scheduling are used to ensure QoS limitations. The NIC driver, a piece of code that communicates with the hardware and delivers packets on the medium, receives the packet once it has been dequeued from the networking layer. In order to prevent excessive queueing, the driver's final queue is usually a FIFO and is governed by a Byte Queue Limit (BQL) [44], [45].

IV. TSQ, TPANDAGGREGATION INTERFERENCE

The model currently in use to depict the RTT and bandwidth of TCP traffic over a typical wired bottleneck, as the one used by Google to create TCP BBR [6], is shown in Figure 2 with solid lines. Three regions are represented by the model: (i) a bandwidth-limited one where the TCP flow reaches the system's bandwidth-delay product (BDP) and increasing the amount of data in flight has the sole effect of increasing the RTT because packets begin to accumulate at the bottleneck link, causing the so-called bufferbloat effect; (ii) an application-limited one where a TCP low can increase the delivery rate without affecting the RTT, (iii) a buffer-limited area, when the bottleneck becomes full and begins to discard packets in order to prevent additional RTT increments. For an extra We refer to [6] for a description of the wired model. This model has been used to illustrate the operating point of various TCP congestion controls. For instance, the model-based variant BBR and the delay-based variant TCP New Vegas operate near point A, while the loss-based variants TCP Cubic and New Reno operate near point C. The same paradigm also applies to wireless bottlenecks that don't use frame aggregation, in which every packet is sent separately.

When we switch to Wi-Fi, which aggregates packets, we lose RTT linearity and bandwidth in the middle of Figure 2, making B the ideal operating point. While loss-based variations function at point D, frame-aggregation enables the highest throughput. Two factors contribute to the non-linear behavior: (i) the delivery rate is not constant as a function of the length of the bottleneck queue, and (ii) the RTT increment is more than just the k packets' transmission delay at the bottleneck queue. A. You may get more information on this effect in [46]. In conclusion, our objective is to simulate the RTT and delivery rate as a function of the aggregate size, which is influenced by TP rate and TSQ.The bottleneck in our system is assumed to be a single active station that transmits aggregates of k packets at a constant bitrate r. The aggregates are actually made up of k packets, including padding and frame overhead, and the total length is $k \cdot l \cdot 8+loh$ bits, where loh is the physical layer overhead and l is the packet size in bits. Additionally, we can model the number of packets at the NIC waiting to be transmitted as a function of the TSQ size (−t) and TP ratio (−p) by concentrating on the upload, where the bottleneck link is directly the sender's NIC:

$k(-t, {}^{1}p) = -t \cdot p \cdot c.$ (1)

In order to make the subject easier to understand, we refer to a steady-state scenario where the NIC queue is backlogged. This enables us to represent t as a static amount expressed in packets and p as a multiplicative factor. In fact, this multiplication is the result of Algorithms 1 and 2 being executed in a cascade. The fraction of the product t·p that is truly backlogged at the NIC is represented by the constant parameter c, which is always in the [0,1] range. In use-cases where

sender nodes may have various hardware or software characteristics, this parameter is crucial for fitting the model. Although the TCP socket has a maximum of p packets that can be queued in the stack, this does not imply that all of these packets will be backlogged at the NIC and prepared for aggregation. Certain packets may still be waiting at the networking levels because of a software bottleneck or excessive congestion, depending on the CPU load or the number of active flows. These specifics, which center on the quantity of the t·p product that is really queued in the NIC and therefore prepared to be a part of an aggregate, are included in parameter c. An aggregate of k(−t, ∑p) packets takes the following amount of time to transmit:
TxTime(k(t,p)) = k(t,p) · l · 8+loh r +toh

where toh is the per-transmission overhead, which includes the average back-off time prior to transmission, the average block acknowledgement time, and the inter-frame spacing. [47] provides a thorough description of toh overhead. This allows us to calculate the predicted effective throughput Thr(k(t,p)), given that there are no mistakes, collisions, or other active stations:
Thr(k(t,p)) = k(t,p) · l · 8 TxTime(k(t,p))

The ath driver will transmit all of the k(−t, −p) packets in a single aggregate if the number of packets enqueued in the driver is less than the maximum aggregation size (mxg). The amount of time required to finish the transmission is According to the steady-state hypothesis, TxTime(k(−t,−p)). Unfortunately, the delay of only the final packet, pk, which begins to be broadcast with the aggregate as soon as it enters the queue, is represented by the time TxTime(k(t,p)). Rather, as they are part of the same aggregate, the first packet, p1, will have the same transmission delay as pk. It has likewise been waiting in the queue for at leastthe other packages' arrival, awaiting the development of the aggregate [48].A delay of at least 2· TxTime(k(−t,−p)) results from this impact. since it is shown in [46] that, according to our hypothesis, the queueing time for aggregate creation is equivalent to the transmission delay. We calculate the RTT that our system experiences as the total of the network's RTTbase plus the contribution imposed by creating the bottleneck queue through queueing delay and transmission delay, taking into account the insignificant ACKs delay. With a maximum aggregation size of mxg, the maximum RTT(k(t,p)) encountered in a queue of k(t,p) packets is:

RTT(k(t,p)) = RTTbase+2·TxTime(k(t,p)) RTTbase+ k(t,p) if k(t,p)≤mxg mxg TxTime(mxg) otherwise.

Indeed, there will be a k(−t, −p) (4) mxg integer if k(−t, −p) >mxg.

aggregates the queueing delay of the remaining packets that are awaiting transmission, along with TxTime(mxg) packets. The remaining packets in the queue will be combined with the newly incoming packets during the transmission of the integer aggregates to create an aggregate of mxg size, which has a queueing delay equal to TxTime(mxg) [46]. As a result, RTTbase + k mxg TxTime(mxg) was the RTT experienced by k(−t, ¹p) > mxg packets. In a similar vein, the Wi-Fi bottleneck bandwidth is defined as follows:

BW(k)= Thr(k(−t, ¹p)) if k(−t, ¹p) ≤ mxg Thr(mxg) otherwise.

As we wrap up the model definition, it's crucial to note that when we impose a static maximum aggregation size of one packet (mxg = 1), we revert to a wireless interface without enabled frame-aggregation mechanism. This is the situation with the solid lines produced using this method in Figure 2. The model without frame-aggregation has been examined in [46] for more information.

TESTBED:

This section explains our testbed, which is shown in Figure 3. Every test includes a client, a server, and the access point that connects the two via an Ethernet connection and a Wi-Fi connection, respectively. Every node uses the 5.4 kernel version of the Arch Linux distribution.
With a desktop or laptop linked to a Wi-Fi access point via the IEEE 802.11n or IEEE 802.11ac standard, this testbed simulates a typical home or office connection. Gigabit Ethernet interfaces are used to connect the remainder of the network, so they are not the bottleneck, hence they have no impact on our tests. PCIe Atheros chipsets that are compatible with the ath9k and ath10k open drivers provide wireless connection. The customer employs various TCP congestion control techniques (discussed in Section III and Table II) and has the ability to establish various TP rates and

TSQ limitations [8]. We modified the kernel to reveal the TSQ core parameters in order to circumvent the rigid standard behavior of TSQ. This allowed us to modify or disable the TSQ logic, as well as impose static TSQ sizes defined in bytes or packets. This solution is known as Controlled TSQ (CoTSQ) [42]. The CoTSQ patch permits altering the maximum amount of data that can be queued at the present rate, whereas normal TSQ permits each socket to enqueue "1 ms of data." the current rate based on several time periods. This creates a dynamic constraint—that is, it automatically adjusts the number of bytes to enqueue based on the current rate—by limiting the amount of data in the stack as a function of the ms parameter. Because the TSQ size is controlled at the kernel level as a bits shift operation (Algorithm 2) and power of two numbers are favored, we employ values of 1 (standard TSQ), 2, 4, 8, 16, and 32 ms in this research. The TP rate is the other crucial parameter that is presented and examined in this work.Given that BBR does not respond to any changes to the present We patched BBR itself, revealing the internal TP Gain variable, which is the normal pacing setting for Linux systems. We chose three potential pacing rates and gave this modified version the moniker BBR+. These have been applied to both BBR+ and the conventional algorithm. These pacing rates are called 1p, 2p, and 3p. 1p is the basic pacing rate that all TCP variations utilize, 2p doubles the values, and so on. You may read more about our BBR+ patch in [42]. We set up our test computers in accordance with the bufferbloat community's best practice document [49] to prevent the most frequent errors in testing. Next, we turned off every hardware offload capability (such TSO/GSO). All of these modifications work to lessen delays that aren't caused by the algorithms themselves. The Flent [50] tool, a versatile and open network tester that enables the management of various traffic typologies and the auto-collection of numerous performance outcomes, was used to organize all of the tests described in this research.

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

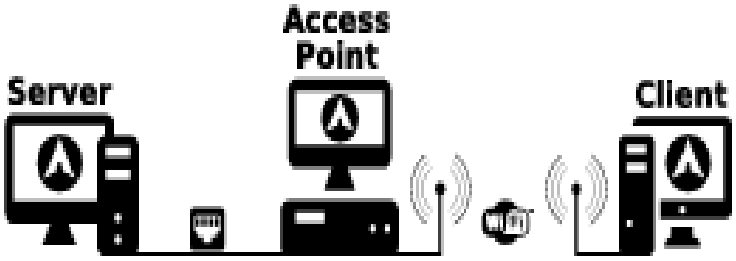


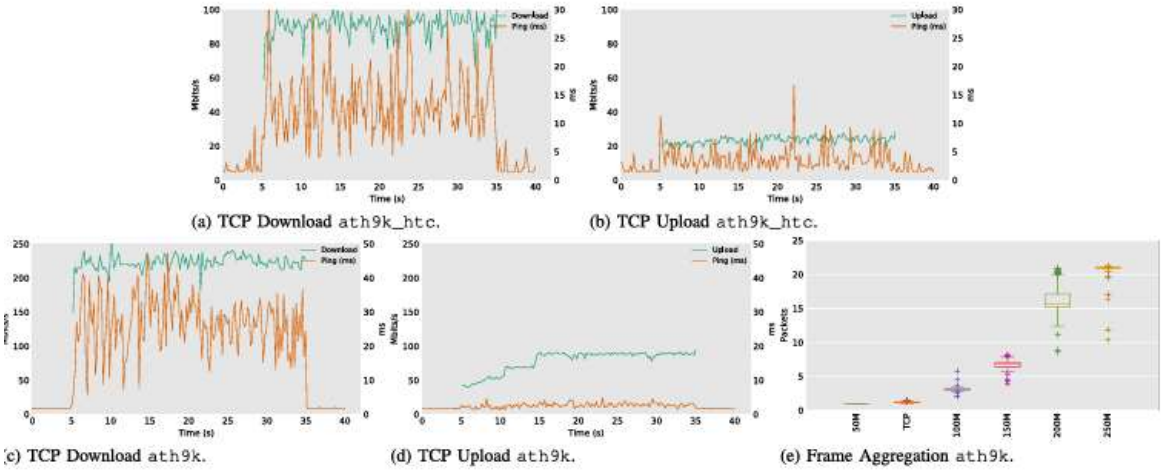
Fig. 3. Physical testbed.

Because it depicts the worst-case scenario for a single TCP upload on a Wi-Fi network, the testbed shown in Figure 3 is straightforward but efficient. bottleneck, where the available throughput is maximized by using frame-aggregation. Our repository contains a variety of situations [42]; in any case, improving the testbed just makes the TCP task easier. For instance, TCP would benefit from more clients as each node would have less bandwidth available. Moving the server can also be taken into account; in fact, moving to a remote bottleneck or raising the RTT would both close the gap to the ideal throughput. Additionally, the performance of various TCP protocols would be equalized by shifting the bottleneck to the wired side, lowering the Ethernet bandwidth, or adjusting the queueing rules. algorithms concerning TP and TSQ, as there would be no involvement with Wi-Fi frame aggregation.

RESULTS:

A. The Reasons for the Work

We demonstrate the TCP upload inefficiency across IEEE 802.11n channels at the beginning of the numerical results section. under the conditions stated in Section III. We will now refer to TCP upload a TCP stream from Client C to Server S and TCP download a TCP stream from S to C with reference to Figure 3. The results of a Cubic download and upload, respectively, utilizing basic USB dongles with Atheros AR9271 1 × 1 MIMO chipsets and ath9k_htc as the driver, are shown in Figures 4a and 4b.



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

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

Despite the hardware's very small capacity, it is instantly apparent that there is a significant difference between the upload and download streams in in terms of TCP goodput, which is expressed in Mbit/s. The highest TCP goodput is nearly 100 Mbit/s due to the dongles' 150 Mbit/s Wi-Fi bandwidth. While the TCP upload cannot achieve goodput values more than 30 Mbit/s, the TCP download goodput approaches ideal and nearly consistent values of 95 Mbit/s. We performed the experiment using PCIe Atheros AR9580 3 × 3 MIMO devices with the ath9k driver in order to fully comprehend the cause

of this imbalance. Apart from the ability to record Wi-Fi statistics like the frame aggregation size is also made possible by the varied hardware. Only ath9k is capable of gathering Wi-Fi statistics data; ath9k_htc and ath10k carry out rate control functions in firmware and do not permit the collection of such data. Figures 4c and 4d show that altering the hardware does not provide different results, even if it is feasible to attain greater goodput values because of the 3 × 3 devices' higher maximum bitrate of 300 Mbit/s. The maximum TCP goodput attained in the download exceeds 200 Mbit/s, while the TCP goodput attained in the upload struggles to reach a value near 100 Mbit/s. This imbalance continues. We experimented with various I perfer instances in upload by substituting UDP for TCP and gradually raising the bandwidth load from 50 to 250 Mbit/s. As a result, UDP can achieve download and upload speeds of more than 200 Mbit/s. This increases the difference between the goodput of a single TCP upload and a single UDP upload. We next gathered Wi-Fi statistics from both TCP and UDP experiments to finish the image. The average frame aggregate size of the several UDP instances is displayed in Figure 4e in relation to the throughput load and the single TCP upload that, as seen in Figure 4d, approaches 50–100 Mbit/s of goodput. The outcome is evident and demonstrates that whereas an increase in UDP load leads to an increase in frame aggregation and a corresponding increase in goodput, with TCP, the aggregation hardly occurs at all.

From now on, we use ath9k to refer to the tests carried out using PCIe Atheros AR5BHB116 2 ×2 MIMO devices (maximum TCP goodput of 200 Mbit/s) and ath10k to the tests carried out using PCIe Qualcomm QCA6178 2 × 2 MIMO devices (400 Mbit/s maximum TCP good put. We modify the TCP congestion control in order to continue our analysis. We present the outcomes of seven chosen variations: Cubic, BBR, New Vegas, YeAH, New Reno, High Speed, and BBR version 2. These are the most typical set of TCPs for our experiments, as stated in Section III. We also evaluated every other Linux default version, and the outcomes are available in [42]. None of the TCP variations can achieve the ideal TCP upload goodput of 200 Mbit/s, as Figure 5a illustrates. A. In other words, because the various slow-start algorithms interfere with the TSQ logic in different ways, TCP Cubic performs worse than TCP New Reno. Additionally Figure 12 will provide a detailed description of the effects of the various TCP slow-start techniques. In contrast, Figure 5b shows the outcome of four concurrently running TCP uploads. The first thing to observe is an increase in cumulative goodput. by boosting the quantity of active TCP flows, especially for the first three TCP types of New Vegas, BBR, BBR v2, and Cubic. Increasing the quantity of TCP uploads has little impact and doesn't significantly alter good put for the remaining variations. There is a glaring inefficiency given that the ideal goodput value is approximately 200 Mbit/s. The TSQ algorithm is the cause of this, disrupts the frame aggregation mechanism regardless of the TCP variant, making it more difficult for the TCP algorithm to queue the quantity of packets required to create larger frame aggregates in order to achieve better goodput. Figure 5b recorded somewhat higher goodput values than Figure 5a because the presence of several TCP flows somewhat increases the number of accessible packets.

B. Data vs. Model

We conduct an experiment using a single TCP upload with the ath9k_htc devices as a function of TSQ size ^{1}t and TP ratio ^{1}p in order to verify the model of Section IV. TCP is our choice. Cubic for this test because, according to Equation 1, its loss-based behaviour permits adjusting the number of NIC packets through t, p, and c, hence controlling the point C of Figure 2. By using our test to determine our sender characteristics for this experiment, the parameter c has been empirically determined.

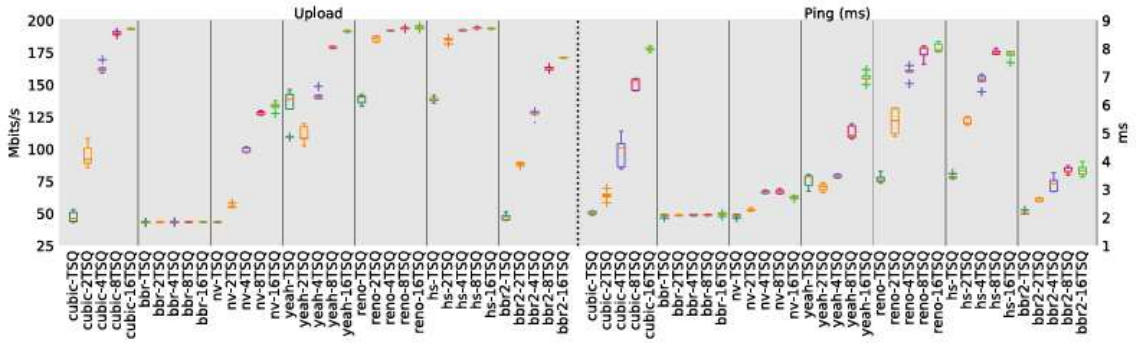
Additionally, ath9k_htc is chosen for the same reason; otherwise, the driver's inbuilt FQ-CoDel would not permit ath9k. us to raise the RTT by more than 5 ms based on the in-flight data. The experiment's findings are shown in Figure 6, where the RTT of the TPC flow is correlated with Equation 4 on the right and the throughput of the TCP Cubic flow with Equation 3 on the left. The parameters listed in Table I have been used to calculate the model's curves. It is evident that the data gathered closely reflects the Section IV model. For two reasons, we employed the packet-size version of our TSQ fix. First, it enables us to use fine-grained x-axes for the collected data (the user may only choose discrete ms values that follow the power of two with the traditional definition of TSQ based on the latency). Second, because the ms-version of the TSQ patch is rate-dependent, it is more in line with the steady-state assumption of the model. The findings demonstrate the RTT's initial ramp and stairs behaviour as well as the TP rate's multiplying effect on the x-axes, which increases the amount of data in flight with the same TSQ value.

C. TSQ's effects

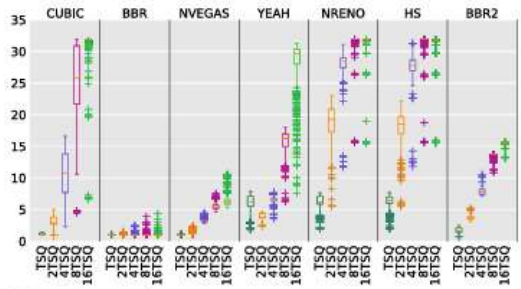
We proceed with our research by utilizing the ms-version of the TSQ patch, which permits each TCP variant to enqueue more than 1 ms of data at the current time by adjusting the TSQ size TCP speed. Specifically, we permit the enqueuing of the equivalent of x ms of data, designating When examining Figure 7, the first thing to note is that, aside from BBR, goodput considerably increases for all the
Several TCP variations have reached 200 Mbit/s. Another thing to note is that, generally speaking, TCP goodput increases as a function of TSQ size. Additionally, an increase in TCP goodput is correlated with an increase in ping recorded latency. YeAH has the best goodput-latency trade off among the TCP algorithms that can achieve the ideal goodput of over 200 Mbit/s, with a latency of 7 ms as opposed to 8 ms for Cubic, New Reno, and Highspeed. sizes affect each TCP variation. The usual TSQ size of 1 ms, called simply TSQ, was altered to 16 ms, called 16TSQ. In response to the TSQ increment, New Vegas showed an odd behaviour: it increased the TCP goodput from less than 50 Mbit/t to nearly 150 Mbit/s while keeping the latency consistently around 3 ms. As previously stated, the only the key figure of worth in this case is BBR, which does not react to TSQ fluctuations because of its programmed behaviour to regulate the latency (down to 2 ms, even easing the TSQ constraints). Conversely, the new BBR v2 version can enable throughput expansion as a function of TSQ size, reaching nearly ideal levels between YeAH and New Vegas in terms of throughput, with New Vegas having the best latency.

In order to fully comprehend the effects of the TSQ size increase, we now turn to Figure 8, which presents information on the frame aggregation size and the TCP RTT from the same experiment seen in Figure 7. As discussed in Section IV, TCP and TSQ combinations that approach the maximum aggregation size achieve the best goodput. Every TCP variant exhibits distinct characteristics, with the exception of New Reno and High Speed (as well as other unreported variants like Scalable, Illinois, Westwood+, and Hybla), which respond to the TSQ variation quite similarly. A. Cubic aggregates less as a function of the TSQ reporting lower RTT values than New Reno. Figure 8b shows something is absent from Figures 7 and 8a, including the subtle distinction between New Reno with 8TSQ and 16TSQ. Relaxing the TSQ limits from 8TSQ to 16TSQ results in an increase in the TCP RTT from 12 to 18 ms, even with the same aggregation size of 32, goodput of over 200 Mbit/s, and ping latency of 8 ms. This also applies to HighSpeed. This is due to the fact that the TSQ mechanism permits more data to be pushed down the stack, filling the buffers and creating a queueing delay that raises the

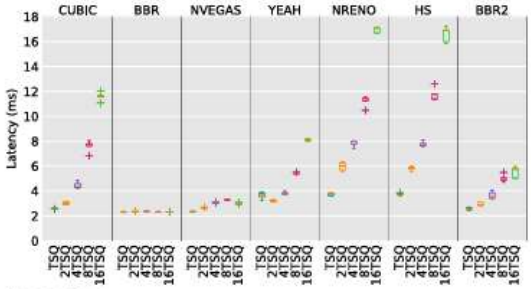
TCP RTT values. Once more, YeAH achieved the optimum trade-off between aggregation size and TCP RTT; it can contain the TCP RTT at 8 ms while achieving an aggregation average of 30 packets and obtaining goodput values of 200 Mbit/s (Figure 7). The aforementioned New Vegas-related factors are applicable here as well. Additional information about the BBR behavior is provided in Figure 8a. Increasing the TSQ size results in only a few tries to aggregate more (five packets with 16TSQ), while the average BBR aggregation size is firmly steady at 1.



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



a) Frame Aggregation.



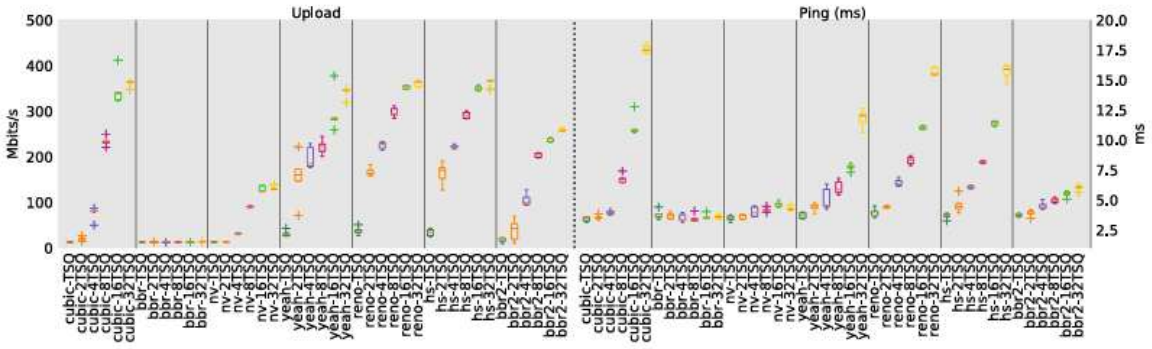
(b) TCP RTT.

This most likely occurs during the BBR probe phases, however the impacts are promptly mitigated by the BBR drain phases, where the latency is frozen as the most crucial aspect to manage. The evolution of BBR, on the other hand, confirms its good performance with these figures as well; BBR v2 permits frame aggregation while maintaining very low latencies, with a good trade off comparable to YeAH. The only notable difference is at 16TSQ, where YeAH nearly reaches optimal throughput while BBR v2 reaches 175 Mbit/s based on the earlier results.

Finally, Figure 8a allows us to clarify the various TCP goodput values displayed in Figure 5a. YeAH, High Speed, and New Reno can all aggregate when standard TSQ is in place. between 6 and 7 packets, whereas New Vegas, Cubic, BBR, and BBR v2 do not aggregate at all. The TCP variant code's usage of time and its interaction with the TP module, which will be examined in the section that follows, are the causes of this inequality. As a result, New Reno, YeAH, and HighSpeed are more likely to enqueue packet bursts, which makes aggregate building easier and leads to greater TCP goodput values with the regular TSQ. The disadvantage of these TCP variations is that, even with the typical TSQ in place, the TCP RTT is somewhat higher and nearly 4 ms.

D. IEEE 802.11ac

The ath9k driver is being replaced by the ath10k driver as we go from IEEE 802.11n to IEEE 802.11ac technology.It is crucial to note that it is not feasible with Ath10k. to gather data on frame aggregation and any other Wi-Fi statistic because of the locked firmware. However, we can show how the TSQ size affects TCP latency and goodput. Similar to Figure 5, increasing the number of concurrent TCP uploads from 1 to 4 does not address the inefficiency imposed by the standard TSQ size in an IEEE 802.11ac environment. The data pertaining to the same test on the ath10k driver are available in [42] and are not included here due to space constraints.



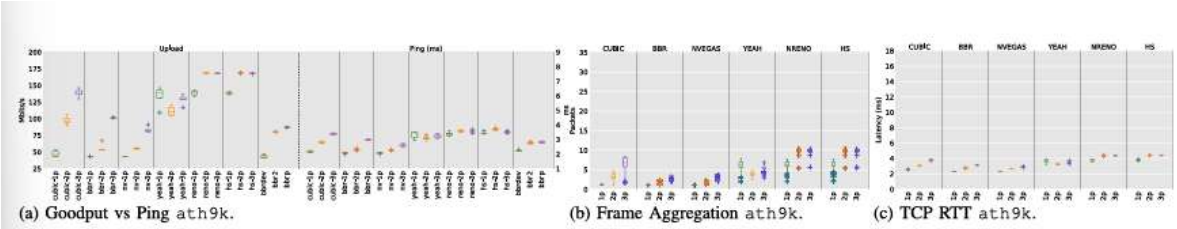
g. 9. One TCP flow in upload: different TCP & TSQ, Goodput vs. Ping, ath10k.

Figure 9 shows the findings about the effect of the TSQ size during a single TCP upload. Here, because IEEE 802.11ac permits a greater maximum bitrate, we also incorporated the 32TSQ results into the plot. Indeed, optimal throughput with ath10k can be reached by allowing more
data to be enqueued with respect to ath9k. The overall pattern is comparable to the equivalent experiment with IEEE 802.11n; HighSpeed and New Reno exhibit similar behaviour, but the latency increases from 11 ms to 16 ms when switching from the former to the latter, reaching 400 Mbit/s with the 16TSQ and 32TSQ configurations. YeAH achieves comparable outcomes, although it is more limited in terms of latency and goodput for the same TSQ sizes. New Vegas responds to the shift in TSQ size with a small increase in latency and a weaker goodput boost of up to 150 Mbit/s. The lowered TSQ limitations continue to have no effect on BBR. In a similar vein, BBR v2 places itself between YeAH and New Vegas in terms of throughput, albeit with somewhat higher latency values than New Vegas. Lastly, Cubic approaches the ideal goodput with 16TSQ and 32TSQ setups, whilst New Reno and HighSpeed exhibit comparable outcomes. In conclusion, even though Figure 7 and Figure 9 show comparable patterns, it is evident that IEEE 802.11ac needs to loosen the TSQ.
limitations to increase goodput to a level near its maximum values since these exceed IEEE 802.11n.

E. TP's effects

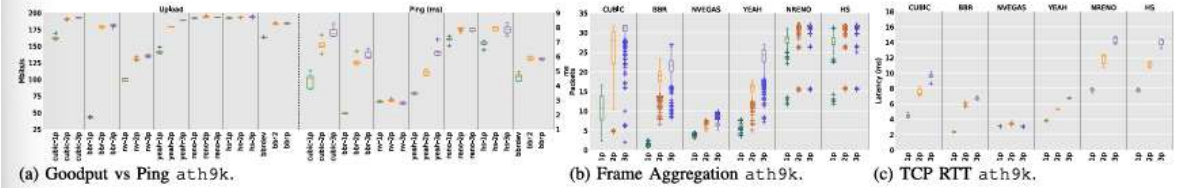
We altered the BBR internal pacing module for this test run so that it would react to the same adjustments that we would apply to the other TCP variants, i.e., when we want the BBR pacing rate to double along with the "global" pacing rate. We include the BBR variations of BBR-DEV, BBR v2, and BBRp in the results for a more equitable comparison because they deal with particular modifications to the BBR pacing system.

We want to demonstrate how TP affects the frame aggregation size and, in turn, the trade-off between latency and goodput using the following series of tests. To separate the pacing effect, we use the usual TSQ and send a single TCP upload with the ath9k setting. The results in terms of goodput vs. latency and frame aggregation size vs. TCP RTT are displayed in Figure 10, which was obtained by changing the TP rate from the standard value denoted by 1p to 2p by doubling it for both the slow start and congestion control phases, and to 3p by triplicating it in the same manner. Even if the standard TSQ restricts the quantity of TCP data that can be queued in the NIC, it is evident that TP plays a crucial function. Excluding for the time being Indeed, depending on the pacing rate, all TCP variations might create larger frame aggregates, which boosts TCP goodput. This is due to the fact that a greater pacing rate increases the likelihood of producing a burst of packets rather than dispersing them over time, which aids in the construction of frame aggregates that produce a higher goodput. Cubic changes from 45 Mbit/s at 1p to 140 Mbit/s at 3p, moving from 1 to 7 packets per aggregate and incurring an additional 1.5 ms delay. BBR and New Vegas exhibit very similar behavior, beginning to aggregate packets and double the goodput, responding to the pacing rate with fewer but still considerable effects, in the BBR scenario, from 40 Mbit/s to 100 Mbit/s. As the TP parameter rises, New Reno and HighSpeed likewise get better. Both create aggregates made up of 10 packets at a pacing rate of 2p, saturating the channel with 175 Mbit/s. YeAH, on the other hand, exhibits an unexpected decline in performance. YeAH's hybrid nature could be one explanation. Regarding the BBR variants, BBR-DEV yields findings that are comparable to normal BBR because it cannot enhance BBR performance when the first hop is wireless, as in our upload experiment.



(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.

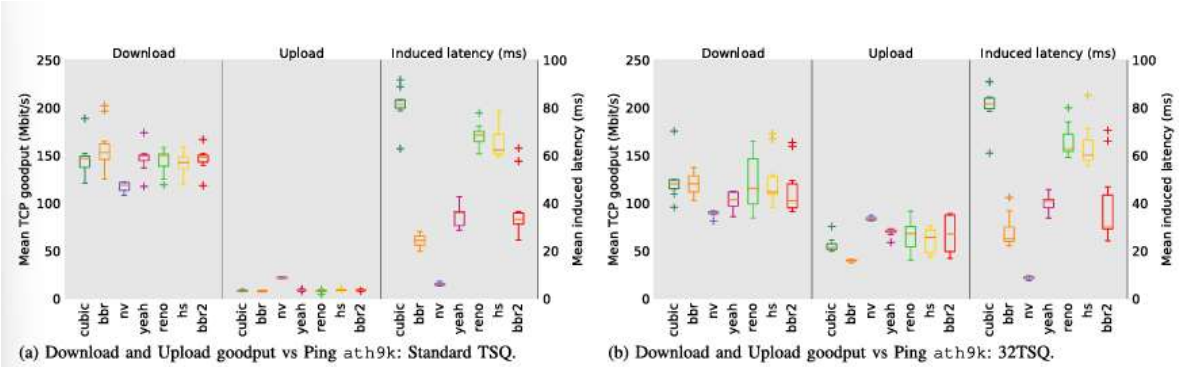
Instead, BBR v2 and BBRp catch up to the loss-based TCP variations, offering throughput and latency comparable to Cubic when paired with a 2p pacing rate.

In Figure 11, we present the same experimental results where we relaxed the TSQ size at 4TSQ, concentrating once more on the effects of various TP rates. We display the performance of 4TSQ since the Linux kernel's default TSQ size for the ath9k driver is this. Due to space limits, further figures displaying various TSQ sizes are not included here but can be obtained in our repository [42].Figure 11 shows that when TSQ is relaxed to 4 ms, Cubic obtains the maximum aggregation size. Additionally, YeAH and BBR respond to the various TP rates more significantly with the 4TSQ setup, nearly achieving the ideal goodput. In order to enable packet aggregation and increase the goodput from less than 50 Mbit/s to values greater than 175 Mbit/s, the TP increment is essential to BBR. In contrast, New Vegas exhibits virtually insignificant performance changes in relation to the TP rate. It is important to see how, when a high TP rate is applied, the TCP RTT of New Reno and HighSpeed begins to significantly grow with just 4TSQ, reaching 14 ms using 3p. The TSQ limitations are relaxed by the favorable reactions of all the BBR variations. They all do, in fact, boost throughput without sacrificing latency. While BBR v2 and BBRp once more record almost optimal throughput values, which are comparable to those recorded by BBR with high pacing rates, BBR-DEV enhances the performance of BBR with standard pacing. This outcome demonstrates how both BBR v2 and BBRp benefit from larger tailored pacing ratios, which support their respective once the TSQ permits it, algorithms search for additional available bandwidth.

The distinct effects of TP on TCP Cubic, on the one hand, and TCP New Reno and HighSpeed, on the other—all loss-based variations—are a significant result of Figures 10 and 11. In Figure 12, we compare the CWNDs of Cubic and New Reno at a function of the pacing ratios in three distinct scenarios: standard TSQ in Figure 12a, 4TSQ in Figure 12b, and with the TSQ mechanism disabled in Figure 12c, respectively, in order to better understand this difference, especially the one shown in Figure 10. This comparison is crucial because when the bottleneck is local, TSQ obscures the behavior of the TCP congestion control mechanism. Put otherwise, if packets build up in the sender NIC, like in our upload tests, Then rather of using the traditional TCP congestion control technique, TSQ limits the CWND by cross-layering interruptions. Figures 12a and 12b show this technique, with extended intervals of several seconds during which the CWND is not updated. intervals of a few seconds during which the CWND is not updated. Since we have shown in Section III how TSQ interferes with the CWND, such interference must be combined with TP. and TP work together to restrict the CWND by defining the maximum number of packets that can be queued. Because it uses a different slow-start algorithm called Hybrid Slow-Start and because TSQ interferes when the bottleneck is local, TCP Cubic behaves differently from TCP New Reno and, generally speaking, from any other loss-based variant. In contrast to New Reno, TCP Cubic must first enter the congestion avoidance phase. This mechanism has two effects. First, only TCP New Reno at slow start can benefit from the rapid growth of CWND since the slow-start pacing rate phase is greater. Second, because the TP rate is lower, TCP Cubic needs more time to raise the CWND during the congestion avoidance phase. In any case, the two figures show the impact of raising the TP rate, which increases the TSQ size at 4 ms by narrowing the difference between Cubic and New Reno. TCP streams are somewhat short-lived due to the hybrid slow-start compromise. In order to wrap off this research, we additionally present the outcomes of the identical experiment with the TSQ mechanism turned off, which is only possible with our patch which are shown in Figures 12c. This figure illustrates how the TCP congestion control

algorithm controls the CWND once more when the TSQ mechanism is disabled. The Cubic and New Reno shapes are easily recognizable and unaffected by TP modifications because, according to Algorithms 1 and 2, TP can only influence the TCP congestion control behavior through the TSQ mechanisms.

The ath10k driver was then used to examine the effect of TCP pacing over IEEE 802.11ac. Even in this instance, TP has a notable effect on performance in a number of situations. We only report TCP goodput, ping latency, and TCP RTT since, as previously mentioned, we are unable to report data regarding frame aggregation owing to driver limitations. Additionally, we extended queue sizes up to 8TSQ, as shown in Figure 14, after testing TP with normal TSQ in Figure 13. This is because the IEEE 802.11ac standard mandates that thanks to our patch, 8TSQ is the default value imposed by ath10kdriver in the current Linux kernel, allowing you to relax the TSQ size more to enjoy the best performance improvements. Where more plots are available, we require [42] readers who are interested in the combination of various TP ratios with various TSQ sizes. Figure 13 illustrates how raising the pacing rate affects certain TCP versions while using conventional TSQ. YeAH, New Reno, and HighSpeed all respond similarly, increasing their goodput from 30 to 150 Mbit/s when the pacing rate is changed from 1p to either 2p or 3p. without appreciable TCP RTT increases or delay. When used with a 3p pacing rate, even Cubic's goodput increases slightly from 25 to 60 Mbit/s. In contrast to the IEEE 802.11n environment, BBR, all BBR variants, and New Vegas do not exhibit any performance changes related to pacing rate while standard TSQ is in effect. Instead, the configuration 8TSQ is displayed in Figure 14; Cubic, YeAH, New Reno, and HighSpeed all get close to the ideal goodput of 400 Mbit/s. For Cubic in this instance, New Reno and HighSpeed, the goodput difference between 2p and 3p is less noticeable; the primary effect is the latency increase with 3p, which reaches levels between 12 and 13 ms. As a function of the TP rate, YeAH enters a reasonable trade-off period between goodput and latency, with the former spanning between 200 and 350 Mbit/s and the latter between 5 and 9 ms. While BRR is getting close to 300 Mbit/s, Vegas is only able to increase its goodput to 200 Mbit/s with no discernible difference between 2p and 3p. The alternative BBR Variants benefit from the TSQ relaxation between 1 and 8 ms: BBR-DEV matches Cubic's performance with regular paging rate, while BBR v2 and BBRp marginally outperform BBR with 2p pacing ratio in terms of throughput with nearly comparable latency, demonstrating how effectively these two BBR variations function in WLAN contexts. Moving from ath9k to ath10k reveals an interesting detail: ping latency and TCP RTT may be clearly distinguished with ath9k, particularly when combined of high pacing rates and big TSQ. As a result, TCP RTT rises greater than ping latency. For Ath10k, the same cannot be stated. The two distinct values of TCP RTT and ping latency with ath10k are consistent across all testing. The two drivers' queueing discipline is the cause. In fact, ath9k adopts distinct queues for TCP packets and ICMP (ping) packets and includes an integrated FQ-CoDel at the driver level, allowing for fine-grained QoS. Thus, When a large number of TCP packets are queued, the queueing delay causes the TCP RTT to rise. On the other hand, the scheduling policy states that the ICMP packets are less impacted by the various queues. This is not the case for ath10k, which does not have FQ-CoDel built into the driver. As a result, all TCP and ICMP packets end up in the same queue, increasing the ping latency and TCP RTT as a function of the queueing delay.



(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.

F. Response in Real Time Under Load

We provide the results of tests conducted in accordance with the Real-time Response Under Load (RRUL) test suite standard to wrap up our experimental review. RRUL is a well-known test suite developed by the Bufferbloat community to examine network performance under high workloads because bufferbloat and other networking issues like congestion and packet loss are easily caused in such situations. By default, the Flent package incorporates the RRUL test, which

comprises of eight bidirectional streams that run against ICMP and UDP traffic (four TCP streams for download and four TCP streams for upload). We set up the RRUL so that the upload and download streams would always be using the same TCP variant.

The results of the average TCP goodput in upload and download, as well as the latency computed on the ping RTT using the ath9k driver and the typical TSQ in operation. The tests employ the same replication and duration parameters as those in the preceding sections. The primary conclusion of these initial testing is that there is a notable imbalance between download and upload streams, with TCP goodput for all examined TCP variants—aside from New Vegas—constrained to 10 Mbit/s in upload but nearly 150 Mbit/s in download. With 125 and 25 Mbit/s for TCP goodput in download and upload, respectively, the latter is actually a little more equitable. The latency, which varies depending on the congestion control technique employed, can be used to distinguish between TCP variations. Cubic has New Reno and HighSpeed have high latencies exceeding 60 ms, whereas the maximum latency is 80 ms. YeAH, BBR, and BBR v2 consistently remain below 35 ms. The best TCP version in terms of latency is New Vegas, which has less than 10 ms. This confirms the algorithm's well-known trait of being less aggressive, which lessens the consequences of congestion. A. The TSQ algorithm itself is one of the causes of the imbalance between the TCP goodput in upload and download. The TCP download in our testbed streams are produced by the server, which is linked to the router and has no restrictions on the low-level aggregation of packets; that is, every TCP variant can achieve the maximum throughput without being constrained by the TSQ method. Conversely, each TCP variation is facing the problem of a restricted frame aggregation with regard to the four TCP upload streams. This is because when the node is wirelessly uploading material to the router, the TSQ mechanism interferes with frame aggregation. The findings are shown in Figure 15b after we adjusted the TSQ size to an equivalent of 32 ms. The global fairness between TCP goodput is a crucial observation. The TSQ limit is relaxed at 32 ms in both download and upload, with varying outcomes for each TCP version. This is a substantial figure when compared to the values we evaluated while the single TCP upload streams were in place. In the 32TSQ setup, nearly all TCP variations show an increase in TCP upload goodput and a decrease in TCP download goodput while maintaining a constant global latency. It is evident from looking at Figure 15 as a whole that New Vegas is the only TCP variation that can achieve almost ideal fairness between download and upload streams. with no distinction between upload and download throughput. Additionally, New Vegas continues to promise ping RTTs of less than 10 ms, making it the best in terms of latency. In terms of fairness, TCP YeAH and BBR v2 differ by about 30 Mbit/s between download and upload goodput. The poorest variation is BBR, which cannot guarantee sufficient upload throughput in a scenario with a standard pacing rate.

Using the standard, double, and triple TP rates of the previous experiment with the 32TSQ configuration, we also examined the effect of TP on the TCP upload streams. 1p, 2p, and 3p in that order. Figure 16 presents the findings. The upload streams' TP rate has a small effect on the RRUL experiment. By raising the TP rate for all TCP variations, the goodput disparity between the TCP download and upload is decreased to a few Mbit/s. Instead, TP has very little effect on ping latency, with only slight increases as a function of TP rate. insignificant, with minuscule variations based on the TP rate. Given the upload goodput, BBR is the only TCP variant that gains from the pacing increase. the ineffectiveness of BBR non a Wi-Fi station's upload path. In conclusion, we report BBRp and BBR v2 in this scenario involving different pacing ratios, with similar results in the RRUL test. We also report BBR-DEV, which demonstrates its nature of increasing the goodput only if the source is wired-connected, aggregating efficiently only in the downstream with respect to the upstream.

In summary, the outcomes of our trials may be analyzed in terms of the trade-off between latency and throughput, where TCP, TSQ, TP, and frame aggregation play a crucial part. The model-based world made possible by BBR and, in particular, the current version BBR v2, which offers a remarkable balance between optimal throughput and ideal latency in practically all of the experiments, is the future direction for TCP macroscopic model creation, according to [15]. The rationale is that after the ideal throughput is attained, monitoring the latency is the only method to control it. However, latency cannot be imposed statically. similar to what occurred with regular TSQ, which only permits 1 ms in order to increase network efficiency when frame-aggregation is available. Our findings indicate that the Linux kernel has been altered to allow for the refinement of the TSQ quantity as a function of the Wi-Fi driver parameters. For the Atheros drivers used for IEEE 802.11n and IEEE 802.11ac technologies, respectively, we specifically chose 4TSQ and 8TSQ.

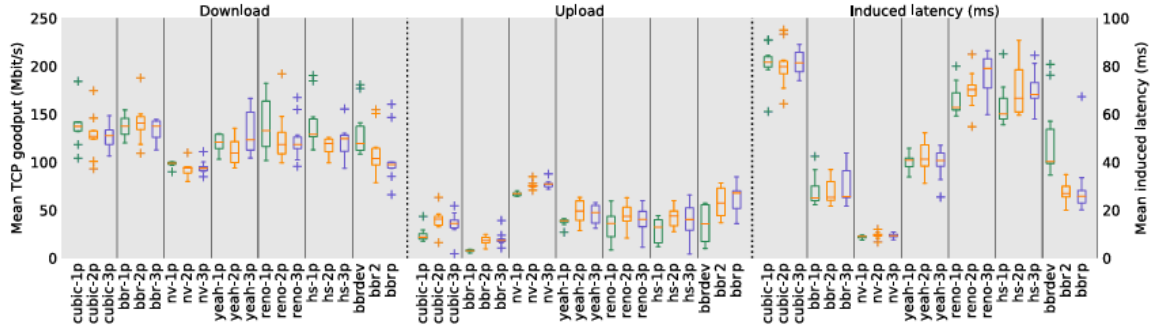


Fig. 16. RRUL test with 32TSQ: `ath9k` varying TP.

CONCLUSION:

In order to increase spectrum efficiency and achieve faster speeds, the most recent WiFi standards heavily rely on frame aggregation. However, higher levels have observed the introduction of methods to ensure better use of the transmission channels while controlling latency. In this report, we examined multiple figures of merit in conjunction with different TCP congestion control methods to determine the potential effects of TSQ and TP on network performance over IEEE 802.11n and IEEE 802.11ac channels. First, it has been demonstrated that TSQ may seriously impair the frame aggregation methods employed by the aforementioned wireless standards, leading to a markedly decreased goodput. To fix the problem, we created and tested a patch that allows the default TSQ size to be adjusted. This change is currently part of the Linux kernel mainline. Numerous TSQ sizes, TCP congestion controls, and wireless technologies have all been extensively tested. All TCP variants—aside from BBR—respond well to larger TSQ sizes at the expense of higher latencies, but to differing degrees. Although BBR does provide some improvements in the frame aggregation size, these improvements are constrained and immediately hindered by the algorithm drain stages. Additionally, the effects of various TP rates have been separated out and examined, improving how setting local restrictions on the volume of data that can be queued also affects the size of the congestion window determined by the TCP congestion controls. When the bottleneck is local (such as the hybrid slow-start), TSQ and TP interaction interferes with TCP congestion control algorithms; this research can aid in the design of new congestion controls that could take this into consideration. All TCP congestion controllers, with the exception of BBR and YeAH, permit the formation of larger frame aggregates when the pacing rate rises, leading to increased goodput. We changed BBR to follow the system TP rate since it incorporates its own TP algorithm. With this modification, BBR exhibits outcomes comparable to those of New Vegas, responding with restricted but substantial benefits to an increase in pace rate. Then, variations in TP rates have been examined alongside variations in TSQ sizes. The findings show that they typically permit higher frame aggregation, which in turn increases latency and goodput. The latter's growth is nonetheless kept within reasonable bounds for the majority of applications. Lastly, tests from the RRUL test suite have been run to evaluate the fairness of various congestion measures against a range of TSQ sizes and TP rates, based on the basic principle of using wired and wireless connections for upload and download transfers, respectively. Findings show that while modifications in TSQ size may greatly improve it, variations in TP rates provide nearly insignificant benefits, even when the fairness unbalances favor wired connections as expected.

REFERENCES

[1] "A tutorial on IEEE 802.11ax high efficiency WLANs," IEEE Commun. Surveys Tuts., vol. 21, no. 1 pp. 197–216, 1st Quart., 2019; E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi.
[2] B. Bellalta, "IEEE 802.11ax: High-efficiency WLANs," IEEE Wireless Communication Magazine, vol. 23, no. 1, pp. 38–46, February 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 Proceedings of the Fifth International Symposium on Computer Networks (CANDAR), November 2017, pp. 216–222.
[4] S. Das, P. Kar, and S. Barman, "Analysis of IEEE 802.11 WLAN frame aggregation under various network conditions," in Proceedings of the International Conference on Wireless Communication and Signal Processing Networks (WiSPNET), March 2017, pages 1240–1245.

[5] "Bufferbloat: Dark buffers in the internet," J. Gettys and K. Nichols, Queue, vol. 9, no. 11, p. 40, Nov. 2011.
[6] N. Cardwell, V. Jacobson, Y. Cheng, C. S. Gunn, S. H. Yeganeh, "BBR: Congestion-based congestion control" 58–66 in Commun. ACM, vol. 60, no. 2, 2017.
[7] P. McKenney, D. Taht, J. Gettys, E. Dumazet, and T. Hoeiland-Joergensen (2018). [Online] FlowQueue-CoDel. Accessible: 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 Proceedings of the IEEE International Symp. Pers., Indoor Mobile September 2018, Radio Commun. (PIMRC), pp. 1–6.

[9] J. Zhou et al., "Server-side TCP stalls: 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 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, August 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 Proceedings of the Fourth Asia-Pacific Workshop Network, August 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 Annu. Conf. ACM Special Interest Group Data Communic. Appl., Technol., Architectures, Protocols Comput. Commun., July 2020, pp. 632–647.
[14] Y. Xie, F. Yi, and K. Jamieson, "PBE-CC: Congestion control via endpoint-centric, physical-layer bandwidth measurements," in Annual Conference ACM Special Interest Group Data Communication Appl., Technol., Architectures, Protocols Comput. Commun., July 2020, pp. 451–464.

[15] "Deprecating the TCP macroscopic model," M. Mathis and J. Mahdavi, ACM SIGCOMM Comput. Commun. Rev., vol. 49, no. 5, pp. 63–68, Nov. 2019.
[16] "Two-level frame aggregation retransmission scheme design in 802.11n/ac/ad," by X. Qian, B. Wu, and T. Ye Xi'an Dianzi Keji Daxue Xuebao/J. Xidian University, vol. 45, no. 2, pp. 90–96, 2018.

[17] "Throughput comparison between the new Hew 802.11ax standard and 802.11n/ac standards in selected distance windows," A. Masiukiewicz, International Journal of Electronic Telecommunication, 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 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 IEEE Conf. Comput. Commun. (INFOCOM), April 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] IEEE/ACM Trans. Netw., vol. 19, no. 4, pp. 975–988, August 2011; S. M. ElRakabawy and C. Lindemann, "A practical adaptive pacing scheme for TCP in multihop wireless networks."
[22] Flow-aware adaptive pacing to reduce TCP incast in data center networks, S. Zou, J. Huang, Y. Zhou, J. Wang, and T. He, Proc. Int. Conf. Distrib. Comput. Syst., 2017, pp. 2119–2124.

[23] "Extensive simulation analysis of TCP variants for wireless communication," A. Pande and S. Devane, 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 Proceedings of the 2nd International Conference on Communication Electron Systems (ICCES), October 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 Proceedings of the 14th International Conference on Telecommunication (ConTEL), June 2017, pp. 75–82.
[26] "A survey of congestion control mechanisms in Linux TCP," C. Callegari, S. Giordano, M. Pagano, and T. Pepe, Commun. Comput. Inf. Sci., vol. 279, pp. 28–42, Oct. 2014.

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

[28] "Measurement analysis of TCP congestion control algorithms in LTE uplink," by A. Parichehreh, S. Alfredsson, and A. Brunstrom, in Proc. Netw. Traffic Meas. Anal. Conf. (TMA), June 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 Communication Magazine, vol. 56, no. 3, pp. 172–179, March 2018.
[30] "TCP fairness among modern TCP congestion control algorithms including TCP BBR," by K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda, and S. Yamaguchi, in Proceedings of the 7th International Conference on Cloud Computing Netw. (CloudNet), October 2018, pp. 1-4.

[31] Y. Zhang, L. Cui, and F. P. Tso, "Modest BBR: Enabling better fairness for BBR congestion control," in IEEE Symp. Comput. Commun. (ISCC), June 2018, pp. 646–651.

[32] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cycle and divergence of performance on TCP BBR," in 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 IEEE 25th Int. Conf. Netw. Protocols (ICNP), October 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 Proceedings of IEEE GlobeCOM, November 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 Proceedings of the 14th International Conference on e-Business Telecommunication, 2017, pp. 89–94.
[36] Cardwell, N. (April 2018). A Linux TCP BBR patch to reduce queuing delays and increase WiFi throughput. RFC. [Online]. Accessible: https://groups.google.com/forum/#!topic/bbr-dev/8pgyOyUavvY

[37] N. Cardwell, "BBR V2: A model-based congestion control," in Proceedings of the ICCRG IETF Meeting, March 2019, p. 36. [Online]. Accessible: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg and %update-on-bbr-00

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

[39] "CUBIC: A new TCP-friendly high-speed TCPvariant," S. Ha, I. Rhee, and L. Xu, ACM SIGOPS Operat. Syst. Rev., vol. 42, no. 5, pp. 64–74, 2008.
[40] In Proc. Int. Conf. Netw. Protocols, 2000, pp. 177–186, 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."

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

[42] January 2021. Tests, source scripts, and Linux kernel patches. [Online]. http://netlab.ing.unimo.it/sw/sourceton.patch is accessible.

[43] YeAH-TCP: Another high-speed TCP, A. Baiocchi, A. P. Castellani, and F. Vacirca, Proc. PFLDnet (ISI), Feb. 2007, pp. 37–42.

[44] Corbet, J. (August 2011). Limits on Network Transmit Queues. LWN article. [Online]. https://lwn.net/Articles/454390/ is accessible.
[45] "Effectiveness of BQL congestion control under high bandwidth-delay product network conditions," by N. Mareev, D. Kachan, K. Karpov, D. Syzov, and E. Siemens, in Proc. Int. Conf. Appl. Innov. IT, vol. 7, no. 1, 2019, pages 19–22.

[46] C. A. Grazia, "A performance model for Wi-Fi frame aggregation taking latency and throughput into consideration," IEEE Communications Lett., vol. 24, no. 7, July 2020, pp. 1577–1580.

[47] "Throughput estimates for A-MPDU and block ACK schemes using HT-PHY layer," by T. Y. Arif and R. F. Sari J. Comput., vol. 9, no. 3, March 2014, pp. 678–687.

[48] "Quick and plenty: Achieving low delay and high rate in 802.11ac edge networks," by H. Hassani, F. Gringoli, and D. J. Leith 2018, arXiv:1806.07761.
[49] J. Gettys and D. Taht (2014). Best Practices for FQ and CoDel Benchmarking. [Online]. Accessible: http://goo.gl/FpSW5z

[50] T. Høiland-Jørgensen, A. Brunstrom, P. Hurtig, and C. A. Grazia, "Flent: The adaptable network tester, in Proceedings of the 11th EAI International Conference on Performance and Evaluation of Methodologies Tools, ValueTools 2017, Venice, Italy, December 2017, pp. 1–6, doi: 10.1145/3150928.3150957.