Towards 1 ms WLAN Latency

Carlo Augusto Grazia
Department of Engineering *Enzo Ferrari*, University of Modena and Reggio Emilia
via Pietro Vivarelli 10, 41125 Modena, Italy. Email: carloaugusto.grazia@unimore.it

***Abstract*—Recent Linux kernel updates have turned on the light again on the needs to reduce network latency in any environment, in particular in Wireless LAN. After the bufferbloat problem definition, many TCP congestion controls have been developed, different queueing algorithms have been deployed and advanced updates on wireless drivers and TCP modules have been reported as well. The contribute of this paper is to present and analyze through real extensive experiments these advanced new solutions highlighting a configuration able to limit a WLAN latency to a value close to 1ms. This result is one order of magnitude lower than the previous literature study and enables WLAN technology to support ultra-low latency communication.**

***Index Terms*—Latency, TCP, TSQ, URLLC, WLAN.**

I. Introduction

Ultra-Reliable Low-Latency Communication (URLLC) is an essential new feature brought by 5G, with a potential to support a vast set of applications that rely on mission-critical links [1]. URLLC means to reduce end-to-end latency and introduce higher reliability in either legacy systems (e.g., UMTS/WCDMA, LTE-A, WLAN, Bluetooth) or in 5G cellular communications [2]–[6]. One key-feature of URLLC is the diversity. Network diversity can be achieved both through using multiple antennas and by using various communication interfaces and technologies as well. This paper investigates mainly the latency impact of WLAN technology enabling the WLAN role of corner-stone in the URLLC world.

Bufferbloat and recent studies to mitigate congestion have steered the networking community research for the last years. Massive improvements in latency reduction have been registered, in particular for wired links, while WLAN technology has not been advertised properly. On one side there have been driver improvements for wireless interfaces [7], and on the other side novel TCP modules, like TCP Small Queues (TSQ), have never been investigated in the literature, and in particular in WLAN environment. The same missing of investigation holds even for the latest advances in TCP congestion controls like the performance of Google BBR algorithm over WiFi [8].

Latency is the primary concern in many applications and domains, such as vehicular safety in V2X communications [9], ultra-reliable communications for industrial automation and public safety [10] as well as generic Wireless Networked Control Systems (WNCS) [11]. There is broad consensus that the main problem of WiFi is that it cannot guarantee low latency in general environment.

The literature lacks in the latency analysis over WLAN concerning the interaction between the novel solutions adopted by Linux systems. This paper fills this gap by introducing these novelties, together with an experimental evaluation, and proposing which are the best configuration when the goal is to minimize the latency. Results show that generic hardware can hit the so wanted 1ms of network latency in several environmental conditions.

The paper is organized as follows. Section II describes the related work on latency investigation over WLAN technologies, while Section III describes the Linux kernel novelties which are the subject of this paper. In Section IV the real testbed used for our experiment is presented, and the results of our tests are reported in Section V. Finally, Section VI concludes the paper.

II. Related work

The performance of WiFi, concerning latency, has been intensely investigated in the literature in the past years. In these works [12], [13], general purpose WiFi network have been studied, and the WLAN latency has been calculated between 20-25ms as a lower bound, and up to 250ms in a congested environment. These measures were taking into account standard TCP applications with default configuration for the end nodes.

In another work [14], the authors model the WLAN latency considering the applications, the 802.11 protocols typology (e.g., 802.11b and 802.11n) and the number of connected devices in order to also vary the level of congestion. Even this work confirms the results of [12], [13] with the same average lower bound of latency in general scenarios.

Saldana et al. analyzed in [15] the tradeoff between aggregation and latency on 802.11n. In fact, the higher throughput of 802.11n with respect to 802.11b is reached, in particular, by enabling frame aggregation [16]. The throughput increment comes at the cost of a latency increment, due to the larger size of the transmitted frame. The paper estimates this latency cost between 10 and 30ms of extra latency induced.

In an emulation based work [17], the authors created an ad-hoc wireless network and concluded, again, an average latency higher than 20ms in general systems.

An outstanding contribution for Linux systems has been the update of the Atheros driver used by the Linux kernel for wireless networking [7]. In this update, the full queueing layer has been integrated into the kernel driver, in order to optimize the behavior of the system regarding latency and mitigating the bufferbloat effect [18]. The FQ-Codel [19] algorithm has been selected as the default algorithm used by the driver.

This last research, together with other recent solutions like TCP Small Queue [20] and TCP Pacing, have contributed to update the entire Linux kernel and make a general Linux system to perform better in wireless networks, in particular in terms of latency. The reported works that state a WiFi latency lower-bound of 20ms are indeed based on old Linux kernel variants or based on different operative systems.

III. Stack

In this section is described the current Linux kernel for what concerns the TCP/IP stack. In particular, the goal is to present the novel TCP block and the Queueing Layer, which are the key modules in charge of latency reduction and take part of our analysis in the numerical results section.

The snapshot of the up-to-date Linux kernel is reported in Figue 1 and contains, from left to right: the TCP main block, the Queueing Layer block and the Driver block.

The TCP block is divided into sub-models. On top of it, there is the TCP Socket, which is responsible for the ACK logic implementation and manages the physical packets. Each TCP connection has its TCP socket that deals with the standard TCP RFC. The packets move from the socket to the lower layer accordingly to the three fundamental sub-models that are:

- Congestion Control: it defines the congestion control algorithm which implements the core behavior of the TCP variant. The congestion control module decides the TCP sending rate, manages the TCP congestion window and calculates the TCP parameter in case of loss or congestion event. In this paper we considered rate-based variants, BBR [8] and New Vegas [21] as well as eight window-based variants which are Cubic [22], HighSpeed [23], Hybla [24], Illinois [25], New Reno [26], Scalable [27], Westwood [28] and YeAH [29].
- TCP Small Queue (TSQ): it is an algorithm that limits the number of TCP packets that the socket can push down in the TCP/IP stack. It is used to prevent the Bufferbloat phenomena in the sender node, by imposing an upper bound of data in the stack for every single flow. The standard TSQ implementation allows each TCP flow to enqueue 1ms of data at the current sending rate (i.e., the actual amount of data depends on the rate, the higher the sending rate, the higher the upper bound).
- TCP Pacing: it is a solution that helps the lower layers of the stack as well as TSQ. The goal of TCP pacing is to distribute the amount of data to send over an entire RTT, avoiding the formation of bursts in the sender node. This is done defining a “pace”, defined as a percentage of the current sending rate, used by the socket to push the data down in the TCP/IP stack.

After the TCP socket dequeues a packet, it moves to the Queueing Layer of Figure 1. This layer can be a stand-alone layer as in the case of the standard wired Ethernet connection, or can be entirely integrated into the Driver as it happens in the `ath9k` driver. We reported the queueing layer as a separate entity because of the driver used in this paper is the ath9k-htc, the one used by the external USB wireless interface. Reported in Figure 1, there is the standard structure of the FQ-Codel [19] algorithm, which is a multi-queue system that deploys the CoDel AQM in each queue and uses a standard Deficit Round Robin scheduler to decide which is the next packet to dequeue. Thanks to the ath9k-htc and so to the possibility to interact with the queueing layer, in this paper also the queueing system implemented in the sender node is considered for testing purposes.

Continuing with the description of Figure 1, the last block is the Driver. This is a piece of kernel code used to approach the Network Interface Card (NIC) and actually transmit a packet on the medium. Another queue is present in the driver, and it is the last queue visited by a packet before to be transmitted, this last queue is typically a FIFO and is ruled by a Byte Queue Limit (BQL) [30] to avoid excessive queueing. This limit is hard-coded in the kernel, so it has not been investigated in our tests.

IV. Testbed

In this Section is reported the testbed used to carry our experiments. The testbed topology is depicted in Figure 2 where three nodes are involved. Because the latency is the central performance value that we analyze, the testbed has been organized consequently to be as fair as possible. A direct node-to-node communication using 802.11n is not typical, it is more frequent to have a client connected via 802.11n to a central node (router) that then provides connectivity to the rest of the network (e.g., a server). This is why our testbed is composed of three Linux nodes, the wireless client, the router and the wired server. The central router has both the wireless and the wired interfaces, to communicate with the client and the server respectively. All the nodes mount a standard 4.14 Linux kernel with Arch Linux as the OS distribution.

The wireless connectivity is given by external USB dongle mounting the Atheros AR9271 chipset. The client and the router nodes communicate with the Atheros chipset through the ath9k-htc open-source driver.

The end nodes, the client and the server are configured to support ten TCP congestion control algorithms which are BBR, Cubic, HighSpeed, Hybla, Illinois, New Vegas, New Reno, Scalable, Westwood and YeAH. The other fundamental parameter modified on the end nodes is the TSQ limit [31], [32]: it can be the standard dynamic value of 1ms of data at the current rate, it can be expressed statically as a number of bytes to limit each queue to a fixed amount, or it can be entirely disabled meaning that no upper bound is applied to each socket.

Another parameter used for testing purposes it the queueing discipline deployed by the router, and in general by all the nodes for consistency. We considered different queueing discipline available by default in the Linux kernel divided among three families:

- Active Queue Managers (AQMs): we tested the simple FIFO policy as well as two novels AQM algorithms developed after the bufferbloat problem has been defined. These two algorithms are PIE [33] and CoDel [34].
- Packet Schedulers: we also tested three standard packet schedulers as Stochastic Fair Queueing (SFQ), Fair

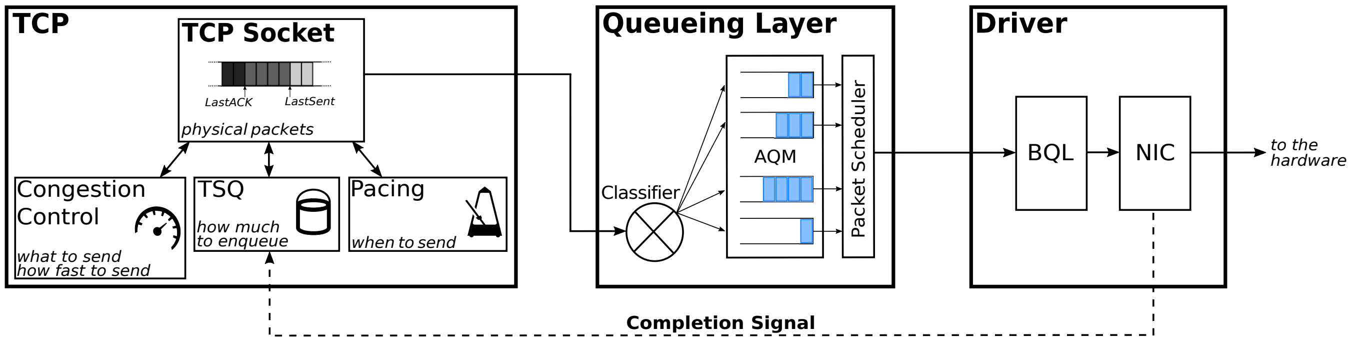


Fig. 1: Linux TCP sender architecture.

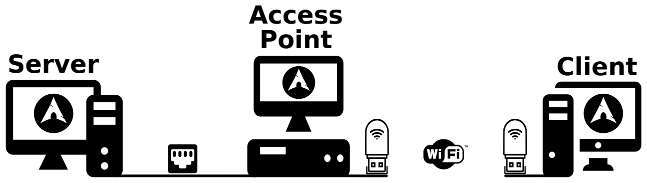


Fig. 2: Physical testbed layout.

Queueing (FQ) and PFIFO. FQ has been developed to enable TCP Pacing for the TCP variants that does not enable it by default, while PFIFO is the basic Linux packet scheduler that deploys three priority queues all served by simple FIFO policy. PFIFO has been the default queueing discipline in many Linux distribution since a few years ago.

- FQ-CoDel: we also tested FQ-CoDel which is the default queueing discipline on Arch Linux currently. It deploys different CoDel queues for each flow and serves each queue in a round-robin fashion through the Deficit Round Robin (DRR) packet scheduler.

The experiments organized for testing the wireless network are all based on the Flent [35] tool. Flent is a flexible network tester that gives the possibility to invoke and monitor easily many traffic typologies. In this paper we always run 30 seconds of TCP traffic; ping traffic is also running simultaneously to each TCP transmission, starting 5 seconds before the TCP connection and ending 5 seconds after, for a total 40 seconds of test for each run. In each experiment, the number of active TCP transmission vary from 1 to 8. We also used another native test of Flent which is the Real-time Response Under Load (RRUL) test; it consists of 4 TCP uploads together with 4 TCP downloads. The goal of this stress-test is to congest the network in both the directions.

The parameters used to configure the testbed of our experiments are summarized in Table I.

V. Results

In this section, the experimental results are reported. The discussion is divided according to the parameter tested for each experiment starting with the TCP congestion control, moving then to the queueing discipline deployed, continuing with the TSQ adopted and concluded with an advanced setup section in which the best combination of the parameters is provided.

TABLE I: Parameters

parameter	value
Kernel version	4.14.14-1
Linux Distribution	Arch Linux
TCP Congestion Control	BBR, Cubic, HighSpeed, Hybla, Illinois, New Vegas, New Reno, Scalable, Westwood, YeAH
TSQ type	TSQ (standard), NoTSQ (disable), BTSQ (Bytes thrashold)
Queueing discipline	FIFO, PIE, CoDel, PFIFO, SFQ, FQ, FQ_Codel
Wireless Chipset	USB Dongle: Atheros AR9271
Wireless Driver	ath9k_htc
Tests	1-8 TCP Uploads/Downloads, RRUL

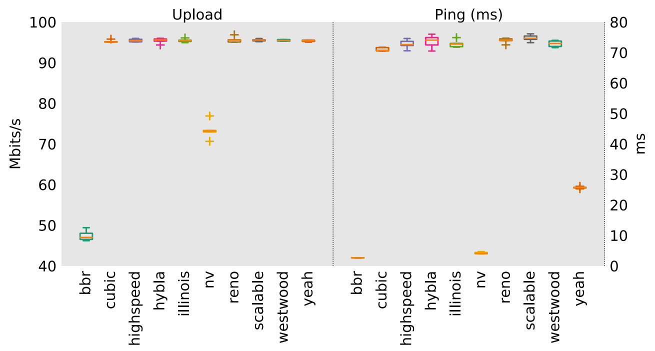


Fig. 3: One TCP flow in upload: Goodput vs Ping.

A. TCP Congestion Control

In the experiment involving different TCP congestion control, we considered a possible old Linux kernel configuration, reflecting the configuration of several related works (e.g., [12]–[14]) where TSQ were not available (TSQ are disabled in this first set of tests) and the default queueing discipline was FIFO. Figure 3 reports the TCP goodput and the latency of a single TCP flow upload from the client to the server, as a function of the congestion control. It is possible to notice how all the TCP variants can saturate the channel capacity reaching almost 100Mbps (which is the theoretical limit in our testbed configuration with the wireless card equipped), the

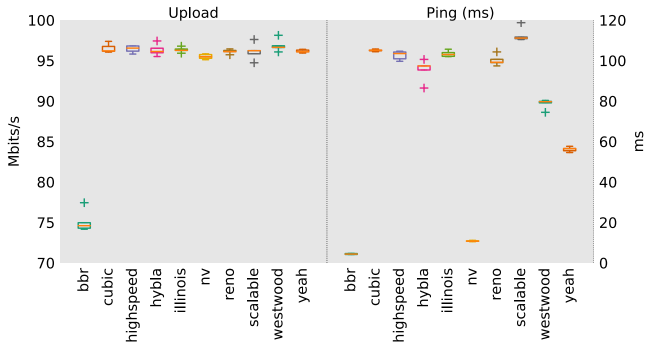


Fig. 4: Four TCP flows in upload: Goodput vs Ping.

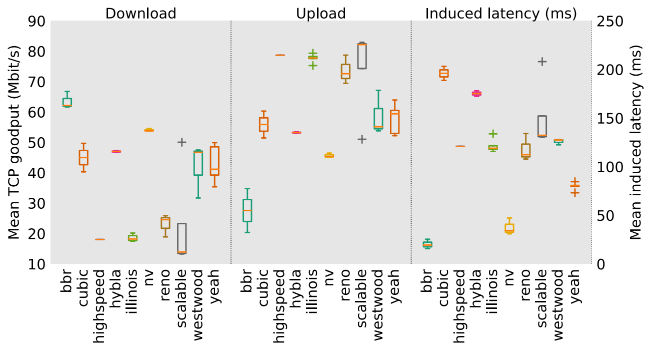


Fig. 5: RRUL test: Goodput vs Ping.

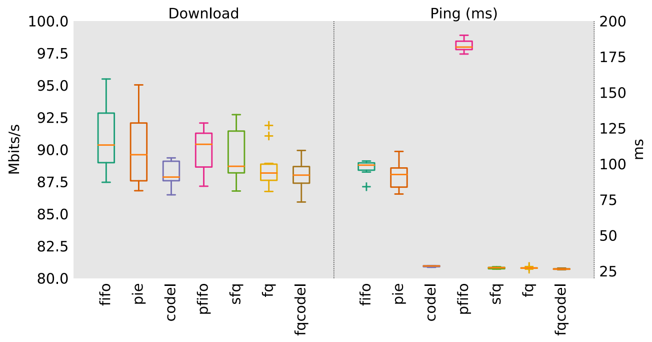


Fig. 6: One TCP flow in download: Goodput vs Ping.

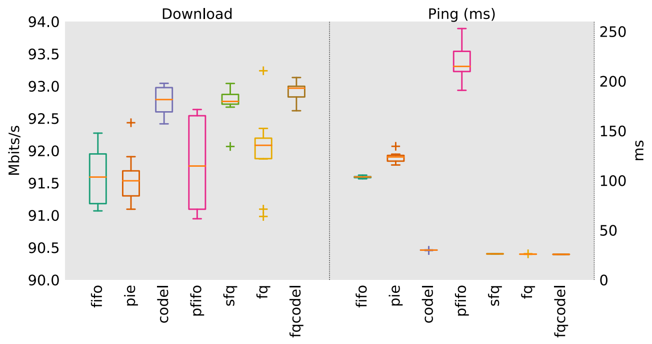


Fig. 7: Four TCP flows in download: Goodput vs Ping.

exceptions are BBR and New Vegas that maintain a goodput of 50 and 70 Mbps respectively. For what concerns the latency, almost all the TCPs are over 70ms, while BBR and New Vegas are under 10ms, thanks also to the uncongested network due to the missing of channel saturation. TCP YeAH reports an interesting result in this experiment: it can simultaneously exploit the channel throughput and limit the latency at 25ms.

We then increase the congestion level moving from 1 to 4 TCP uploads, the results are depicted in Figure 4. Increasing the congestion level has the consequence of increasing the latency for all the window-based TCP variants. Cubic, HighSpeed, Hybla, Illinois, New Reno and Scalable move all over 100ms; Westwood is the only one to increase "only" up to 80ms while YeAH moves from 25 to almost 60ms. Again, the behavior of New Vegas and, in particular, BBR is remarkable: the move to 10 and circa 5ms respectively. The other consideration to collect from this figure is the goodput, in fact, even New Vegas can saturate the channel like the other TCPs while BBR increments the goodput only to 75Mbps.

To conclude the TCP analysis, Figure 5 reports the result of RRUL test. In this case, the upload and download goodput are reported in a separate graph, in order to provide also a fairness metric in terms of the balance between upload and download flows. It is evident the discrepancy between upload and download for many TCPs. The fairest congestion controls are Cubic, Hybla, New Vegas, Westwood and YeAH with only a few Mbps gap between upload and download; HighSpeed, Illinois, New Reno and Scalable are unfair, registering high level of upload (almost 80Mbps) and low level of download (less than 20Mbps), while BBR is the only one with the opposite trend of unfairness: it provides higher download goodput instead of upload one. The reason healds in the BBR nature that has born for data-centers, in fact, the server is attached to the router through a wired connection. The latency in the RRUL test is higher for all the TCP variants. Cubic registered the worst value of 200ms. BBR, New Vegas, and YeAH reported 25, 40 and 80ms respectively, while all the other TCPs are over 100ms.

B. *Queueing Discipline*

We now evaluate the impact of the queueing discipline on the performance of TCP connections. For these experiments, TSQ is still disabled while the TCP congestion control used is the Linux default one: TCP Cubic.

Figure 6 shows the results of one TCP download from the server to the client as a function of the queueing discipline on the router (also the queueing discipline of the nodes is changed accordingly for consistency). It is evident how the FIFO and PFIFO solutions are not adequate to maintain low latency levels, with PFIFO that manifests worse performance because of the larger queue size by default with respect to the simple FIFO. All the smart queueing disciplines as CoDel, SFQ, FQ, and FQ-Codel can contain the latency at 25ms. The only exception is PIE; it suffers the wireless environment and can not reduce the latency registering the same performance of FIFO.

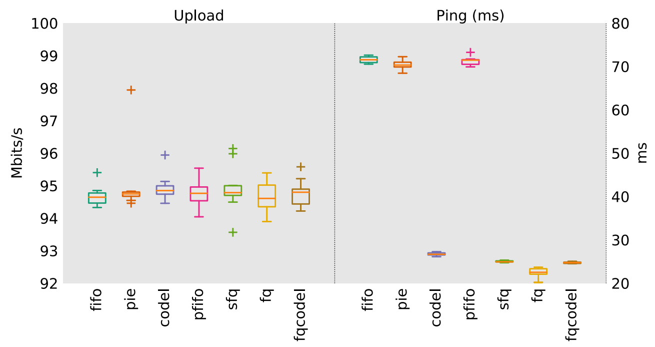


Fig. 8: One TCP flow in upload: Goodput vs Ping.

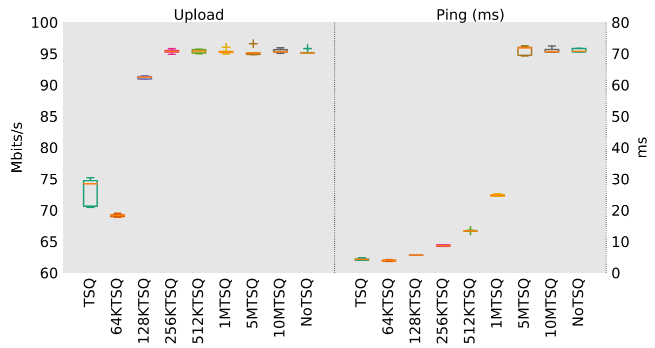


Fig. 9: One TCP flow in upload: Goodput vs Ping.

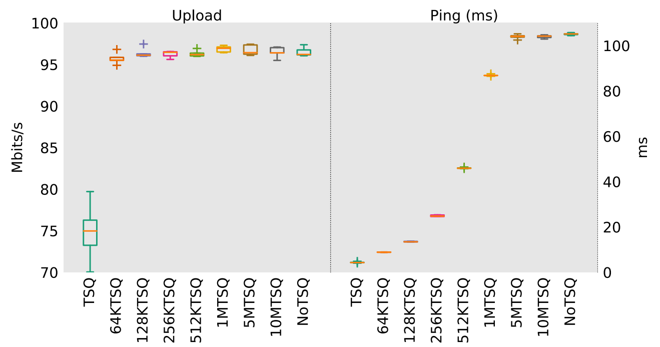


Fig. 10: Four TCP flows in upload: Goodput vs Ping.

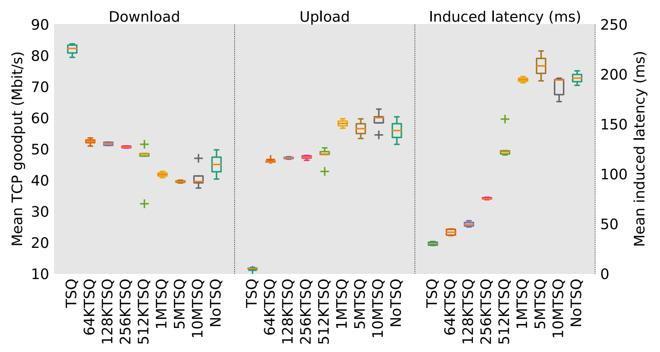


Fig. 11: RRUL test: Goodput vs Ping.

We again stress the network increasing the congestion level from 1 to 4 TCP downloads and report the results in Figure 7. In this case, a slightly different goodput level is reported with CoDel, SQF, and FQ-Codel that registered the best values. The latency values are stable for almost all the queuing disciplines, except for PIE and PFIFO that have a performance degradation suffering from the increased congestion level.

We then took another experiment with 1 TCP upload from the client to the server. The goal is to isolate the backpressure behavior of TCP/IP stack that happens when the sender node is directly attached to a wireless interface, this moves the congestion from the router to the client node and changes the registered results. In fact, Figure 8 shows that in the upload, the performance of PFIFO, FIFO and PIE are the same with worse latency of 70ms. The most relevant thing to report from this experiment is the better performance of FQ with respect to the other queueing disciplines. The reason is the following: TCP Cubic does not enable by default TCP Pacing, and the adoption of FQ enables it. The result is that in the sender node the packets are paced avoiding the burst formation and consequently reducing the latency.

C. *TSQ Type*

Continuing our analysis, here we report the benefit of TSQ. This is a novel module of the Linux kernel and has never been considered in the literature until now.

Figure 9 reports the impact of TSQ on a single TCP upload. The TSQ configuration adopted is the standard one, reported with TSQ, which means that the upper bound limit is dynamic and correspond to 1ms of data at the current TCP rate. The other configurations report a value of Bytes before the TSQ name and correspond to a static upper bound expressed in Byte. The last configuration is named NoTSQ and corresponds to the absence of the upper bound (i.e., TSQ disabled). The TCP used is Cubic, and the queueing discipline on all the nodes is PFIFO. The things to say be looking at Figure 9 are multiple: considering the TCP goodput, the maximum channel capacity is reached if TSQ allows to enqueue more than 128KB of data, allowing a proper frame aggregation for the 802.11n network [15], which corresponds to higher throughput level. The second thing to notice is that, with a single TCP flow, the static value of 64KB can further reduce the latency with respect to the standard TSQ implementation and, in general, the higher the TSQ upper bound, the higher the latency. After 5MB of TSQ upper bound, the latency is stable at 70ms because of the queueing discipline presence.

Increasing the congestion level from 1 to 4 TCP upload lets us discuss another aspect in Figure 10. The latency of the standard TSQ is stable at 5ms while all the other TSQ adopted slightly increase the value up to the worst case of 100ms. From a channel exploitation point of view, even the small static upper-bound of 64KB can saturate the channel, while the standard TSQ implementation maintains the goodput value at 75Mbps.

To conclude the TSQ analysis, the RRUL test in reported in Figure 11. The result here is a direct consequence of the standard TSQ behavior of Figure 10. Because of default

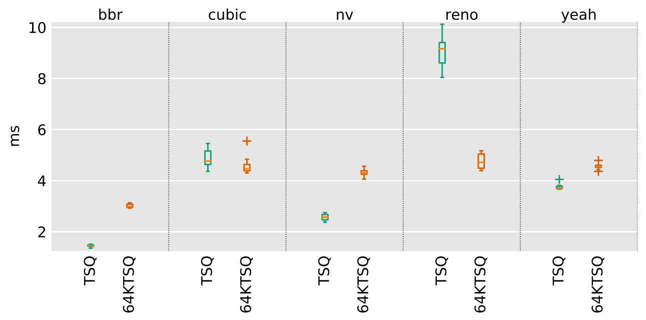


Fig. 12: One TCP flow in upload: only Ping.

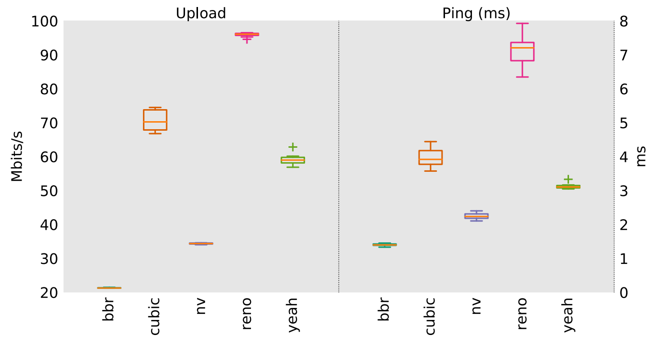


Fig. 13: One TCP flow in upload: Goodput vs Ping.

TSQ is not able to exploit the upload wireless channel, it is overperformed by download TCP where the source is in the server attached to a wired link and the TSQ logic works without the aggregation issue. This is traduced in an extream unfairness between upload and download, while still preserving the lowest latency of circa 25ms. For the other TSQ implementations, the results are fair, and the latency is again a function of the upper-bound level.

D. Advanced setup

This final round of tests proposes a selection of TCP congestion controls and TSQ policies, based on the performance of the previous sections, while the queueing discipline is blessed as FQ-Codel which has provided the best results in the queuing discipline tests. The TCPs selected are BBR, Cubic, New Vegas, Reno and YeAH. The TSQ policies chosen are the standard TSQ one, and the 64KTSQ that forces a static amount of data to enqueue, regardless the rate.

Figure 12 reports the latency results of a single TCP upload grouped by the congestion control adopted. Standard TSQ outperforms 64KBTSQ always, except when TCP Reno is used as congestion control. This is because TCP New Reno does not estimate the bandwidth properly due to its pure window-based nature. Almost the same happens for TCP Cubic where the performance of standard TSQ are the same of 64KBTSQ. The best TCP variant in terms of latency is again BBR with less than 2ms, corresponding to a one-way delay of less than 1ms.

Figure 13 shows the same experiment with only standard TSQ and reports both the goodput and the latency. BBR and New Vegas suffer the channel exploitation registering the lowest values of goodput. YeAH and Cubic have a good compromise with 60 and 70 Mbps as goodput and 3 and 4ms as latency respectively. TCP New Reno fully exploits the bandwidth with almost 100Mbps of goodput but with the highest latency of 7ms.

As the very last experiment we report the RRUL test, with the same advanced setup, in Figure 14a.

Comparing Figure 14a with Figure 14b it is possible to notice how 64KTSQ resolves the fairness problem of standard TSQ (in case of BBR it mitigates the effect without completely fixing it), but increasing the latency of almost 10ms for all the congestion control except BBR. Concerning only latency, TSQ with BBR is the best option even in a congested environment.

VI. Conclusions

We have investigated the latency of a WLAN IEEE 802.11n network by implementing different experiments involving many Linux kernel modules. We focused on different TCP congestion controls as well as on advanced kernel configuration as TSQ and queueing discipline. We demonstrated through a real testbed how a proper configuration can lead to a latency close to 1ms in an uncongested scenario where the novel BBR congestion control is used as TCP variant, the FQ-Codel algorithm is used as queueing discipline, and the standard TSQ module is present. Moreover, the same configuration registered values lower than 10ms in a high-congested network where the RRUL test was in place. In conclusion, this advanced Linux kernel configuration highlights the WLAN technology as a support for ultra-low latency networks, where the delay is the critical performance to consider, guaranteeing a one-way delay lower than 1ms.

References

[1] P. Popovski, J. J. Nielsen, C. Stefanovic, E. d. Carvalho, E. Strom, K. F. Trillingsgaard, A. S. Bana, D. M. Kim, R. Kotaba, J. Park, and R. B. Sorensen, "Wireless access for ultra-reliable low-latency communication: Principles and building blocks," *IEEE Network*, vol. 32, no. 2, pp. 16–23, March 2018.

[2] H. Beyranvand, M. Lévesque, M. Maier, J. A. Salehi, C. Verikoukis, and D. Tipper, "Toward 5G: FiWi enhanced LTE-A hetnets with reliable low-latency fiber backhaul sharing and WiFi offloading," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 690–707, April 2017.

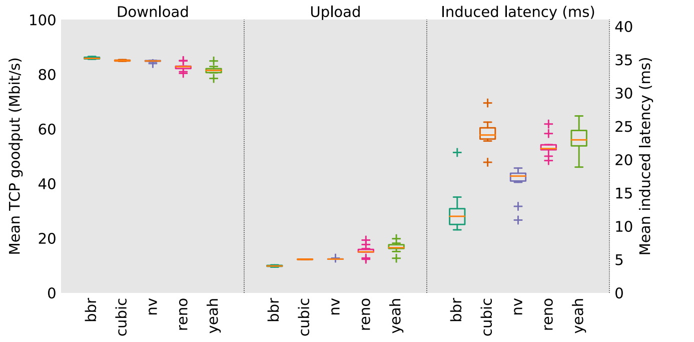
[3] M. Casoni, C. Grazia, and M. Klapez, "An sdn and cps based opportunistic upload splitting for mobile users," in *Proceedings of the EAI International Conference on CYber physiCaL systems, iOt and sensors Networks*, ser. IOT360 Summit '15, 2015.

[4] C. A. Grazia, M. Klapez, N. Patriciello, M. Casoni, H. Gierszal, P. Tyczka, K. Pawlina, A. Amditis, and E. Sdongos, "Performance evaluation and economic modelling of ppdr communication systems," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*, Oct 2015, pp. 75–82.

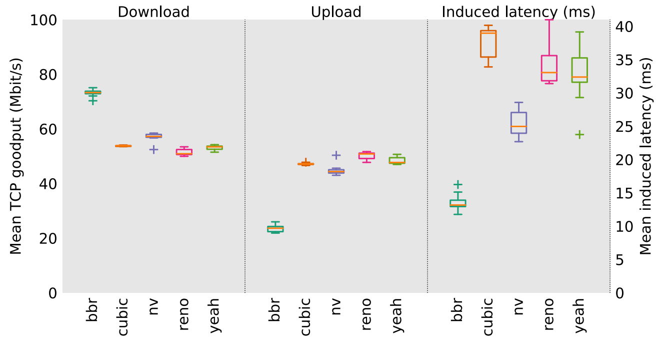
[5] M. Casoni, C. A. Grazia, and M. Klapez, "Enabling resource pooling in wireless networks through software-defined orchestration," in *2016 IEEE ICC Conference*, May 2016, pp. 730–735.

[6] M. Casoni, C. Grazia, and M. Klapez, "Sdn-based resource pooling to provide transparent multi-path communications," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 172–178, 2017.

[7] T. Høiland-Jørgensen, M. Kazior, D. Taht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," 2017, pp. 139–151.



(a) Standard TSQ.



(b) 64KBytes TSQ.

Fig. 14: RRUL: Goodput vs Ping.

[8] N. Cardwell, Y. Cheng, C. Stephen Gunn, S. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[9] A. Balador, A. Bohm, C. Calafate, and J.-C. Cano, "A reliable token-based MAC protocol for V2V communication in urban VANET," 2016.

[10] H. Shariatmadari, S. Iraji, and R. Jantti, "Analysis of transmission methods for ultra-reliable communications," vol. 2015-December, 2015, pp. 2303–2308.

[11] K. Mathews, C. Kramer, and R. Gotzhein, "Token bucket based traffic shaping and monitoring for WLAN-based control systems," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017.

[12] C. Pei, Y. Zhao, L. Yunxin, K. Tan, J. Zhang, Y. Meng, and D. Pei, "Latency-based WiFi congestion control in the air for dense WiFi networks," 2017.

[13] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, and D. Pei, "WiFi can be the weakest link of round trip network latency in the wild," vol. 2016-July, 2016.

[14] K. Suiy, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, and T. Moscibroda, "Characterizing and improving WiFi latency in large-scale operational networks," 2016, pp. 347–360.

[15] J. Saldana, J. Ruiz-Mas, and J. Almodovar, "Frame aggregation in central controlled 802.11 WLANs: The latency versus throughput tradeoff," *IEEE Communications Letters*, vol. 21, no. 11, pp. 2500–2503, 2017.

[16] Y. Kim, S. Choi, K. Jang, and H. Hwang, "Throughput enhancement of IEEE 802.11 WLAN via frame aggregation," vol. 60, no. 4, 2004, pp. 3030–3034.

[17] O. Wellnitz and L. Wolf, "On latency in IEEE 802.11-based wireless ad-hoc networks," 2010, pp. 261–266.

[18] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[19] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-CoDel," https://tools.ietf.org/html/rfc8290, January 2018.

[20] J. Corbet, "TCP small queues," https://lwn.net/Articles/507065/, July 2012.

[21] J. Sing and B. Soh, "TCP New Vegas: Improving the performance of TCP Vegas over high latency links," in *Fourth IEEE ISNCA*, July 2005, pp. 73–82.

[22] S. Ha, I. Rhee, and L. Xu, "Cubic: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS*, vol. 42, no. 5, pp. 64–74, 2008.

[23] S. Floyd, "Limited slow-start for TCP with large congestion windows," 2004.

[24] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004.

[25] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6–7, pp. 417 – 440, 2008.

[26] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," 2004.

[27] R. El Khoury, E. Altman, and R. El Azouzi, "Analysis of scalable TCP congestion control algorithm," *Computer Communications*, vol. 33, no. SUPPL. 1, pp. S41–S49, 2010.

[28] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Com. Com. Review*, vol. 34, no. 2, pp. 25–38, 2004.

[29] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet another highspeed TCP," in *in Proc. PFLDnet, ISI*, February 2007, pp. 37–42.

[30] J. Corbet, "Network transmit queue limits," LWN Article, August 2011. [Online]. Available: https://lwn.net/Articles/454390/

[31] 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," *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 2018.

[32] C. A. Grazia, "On the performance of IEEE 802.11p Outside the Context of a BSS Networks," *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 2018.

[33] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "Pie: A lightweight control scheme to address the bufferbloat problem," in *High Performance Switching and Routing, 2013 IEEE International Conference on*. IEEE, 2013, pp. 148–155.

[34] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[35] T. Høiland-Jørgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," *ValueTools 2017*, 2017.