Accepted for presentation in: 51st IEEE Conference on Local Computer Networks (LCN), Coimbra, Portugal, October 7–9, 2026.

Byte Queue Limits for Linux usbnet Driver: Reducing Host-Side Bufferbloat of USB Ethernet Adapters and Cellular Modems

Simon Schippers, Hendrik Schippers, and Christian Wietfeld
Chair of Communication Networks, TU Dortmund University, 44227 Dortmund, Germany
E-mail: {Simon.Schippers, Hendrik.Schippers, Christian.Wietfeld}@tu-dortmund.de

***Abstract*—Emerging latency-sensitive applications, such as teleoperation, demand minimal delay along the entire network path, including within the host. In Linux, many USB network devices, from Ethernet adapters to 5G modems, rely on the `usbnet` driver, which uses a static, conservatively sized transmit queue. Under load, this queue can cause bufferbloat, resulting in queuing delays of several seconds. In this paper, we analyze the `usbnet` transmit queue, quantify its latency, and propose a kernel patch integrating Byte Queue Limits (BQL) to dynamically size the queue for full link utilization. We evaluate the patch on USB-to-Ethernet adapters, achieving one-way delay reductions of up to 7× without throughput loss. We further analyze three commercial 5G modems and find that they still exhibit substantial internal buffering, yet our patch reduces the median one-way delay by up to 2×. Our patch has been accepted into the mainline Linux kernel, benefiting billions of USB networking devices.**

I. Introduction

Modern applications such as teleoperation of vehicles and industrial robots impose strict uplink latency requirements on the communication stack. For teleoperated driving, requirements range from 100 ms in 5G vehicular studies [1] down to 5 ms in 3GPP specifications for advanced V2X scenarios [2]; real-time robot control demands even tighter budgets of 1–50 ms per control loop [3].

To fulfill those requirements, 5G networks promise Ultra-Reliable Low-Latency Communication (URLLC) [4] with end-to-end latencies in the single-digit millisecond range, but achieving this target requires careful management of packet queues along the entire network path, including in the host operating system. The latency and reliability actually delivered by a wireless link are moreover strongly dependent on the specific application environment and system scale [5], making additional sources of delay particularly harmful.

A well-known obstacle to achieving low-latency networking is bufferbloat [6]: the excessive accumulation of packets in network buffers causes queuing delays that can reach hundreds of milliseconds, rendering latency-sensitive applications unusable. Classical queueing theory already argued that a system should be operated at a load that just saturates its resources, avoiding both under-utilization and standing queues [7]. In practice, however, real network stacks fall short of this ideal and buffers are often conservatively sized for the worst case. The Linux networking stack provides several mechanisms to combat bufferbloat, including BQL at the driver level (see Sec. II), which dynamically sizes the transmit queue to the minimum needed for full link utilization.

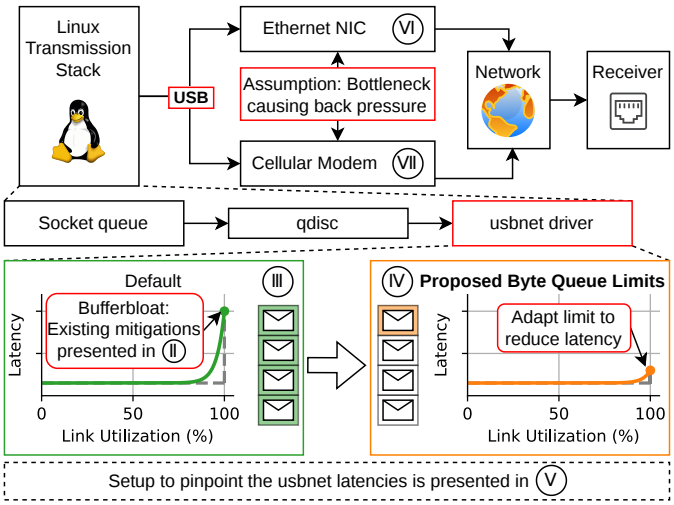

Fig. 1: Overview of the contributions: (II) a related-work analysis, (III) analysis of the `usbnet` transmit queue, (IV) a BQL patch, (V) a targeted measurement setup, and evaluation on (VI) Ethernet and (VII) 5G cellular interfaces.

The `usbnet` driver, used by many USB networking devices, lacked this support and instead relied on a static, conservatively sized queue. Those USB networking devices include 5G modems (commonly also as M.2 modules), USB-to-Ethernet dongles and docking stations, and Android USB tethering.

This paper addresses the bufferbloat problem in the `usbnet` driver by implementing BQL. To the best of our knowledge, BQL has so far only been used in Ethernet chip drivers; this is the first application of BQL to cellular modems.

Since our patch has been accepted into the mainline Linux kernel, the improvements presented in this work will benefit **billions of devices** that depend on commodity USB-attached networking. High-bitrate server and desktop NICs typically use PCIe directly and are outside the scope of this work. Fig. 1 summarizes our contributions:

- II Review related work on bufferbloat mitigation.
- III Analyze `usbnet` transmit-queue latency.
- IV Propose a BQL kernel patch for the driver.
- V Present a test setup, isolating the `usbnet` queuing delay.
- VI Measure latency reduction on USB-to-Ethernet adapters.
- VII Evaluate the impact on three commercial 5G modems.

II. Bufferbloat: Related Work and Technical Background

This section reviews several mitigations of bufferbloat, explains BQL as the mechanism underlying our patch, and introduces the methodology for measuring latency under load.

A. Congestion Signaling in the Transport Layer

At the protocol layer, Explicit Congestion Notification (ECN) [8] allows Active Queue Management (AQM)-enabled routers to signal early congestion by marking, rather than dropping packets, so that Transmission Control Protocol (TCP) can react before buffers overflow. On the sender side, TCP Small Queues (TSQ) [9] limits how many bytes a single TCP flow may keep in flight in the local stack, preventing a single flow from monopolizing the Queueing Discipline (qdisc) and driver queues. More recently, TCP Bottleneck Bandwidth and Round-trip Time (BBR) [10] estimates the bottleneck bandwidth and Round Trip Time (RTT) to pace transmissions just below the link capacity, keeping standing queues empty without relying on loss as a congestion signal. The Low Latency, Low Loss, and Scalable Throughput (L4S) architecture [11] complements these efforts by combining scalable congestion controls with fine-grained ECN marking to achieve consistently low queuing delay.

B. Active Queue Management in the qdisc Layer

Several AQM mechanisms operate at the qdisc (queuing discipline) layer, Linux's per-interface packet scheduler between network stack and device driver. Controlled Delay (CoDel) [12] pioneered queuing-time-based dropping as a parameterless successor to RED and PIE [13], the latter is deployed in DOCSIS cable modems. Fair-queueing variants such as `fq_codel` [12], [14] and shaping-aware CAKE [15] isolate flows into per-flow sub-queues and drop or mark packets as soon as the queuing delay exceeds a limit. While this keeps the qdisc backlog small, its effect is limited to the qdisc itself: Packets that have already been handed down to a driver queue are outside its control, which motivates driver-level mitigations.

In WiFi networks, large per-station queues in firmware and drivers can cause high delay and jitter under load. Linux `mac80211` mitigates this by combining per-station `fq_codel` queueing with airtime-fair scheduling at the access point [16].

C. Backpressure in Software Network Drivers

Software network interfaces suffer from bufferbloat as well. The recently proposed NODROP patch [17] addresses packet drops in the Linux TUN driver by introducing backpressure rather than silently dropping when the internal queue is reduced for low latency, thereby making short driver queues usable for real-time Virtual Private Network (VPN) traffic.

D. Queue Sizing and Parallelism in Hardware Drivers

On hardware Network Interface Cards (NICs), two techniques dominate. Multi-queue support [18] exposes multiple hardware transmit queues, enabling traffic prioritization. Independently, BQL (Sec. II-E) shrinks each queue to the smallest size that still keeps the transmitter busy. Both require explicit driver support; the `usbnet` driver currently offers neither.

E. Byte Queue Limits: Dynamic Queue Sizing Against Bufferbloat

BQL is a Linux kernel mechanism, introduced in 2011 [19] and unchanged in its fundamental design since then, that dynamically adjusts the transmit queue limit of a network device. Its design rests on two principles:

1) **Do not starve the transmitter.** Enough data must be queued so that the hardware is never idle while packets are waiting to be sent.
2) **Do not over-queue.** Enqueuing more data than necessary inflates latency and can even trigger spurious TCP retransmissions for packets that have not yet left the host.

The core objective is therefore to enqueue enough data to prevent starvation, but no more. Unlike traditional device queue limits that count packets, BQL operates in bytes. Because packets are variable-sized, the byte count has a more direct relationship to the time required to drain the queue on the wire.

The implementation builds on the Dynamic Queue Limits (DQL) library, a general-purpose queue-length controller. It requires a driver to call the `netdev_sent_queue` function when bytes are submitted to the hardware and the `netdev_completed_queue` function upon transmission completion processing. After each completion, the DQL algorithm decides whether the byte limit should be raised (if the transmitter was starved) or lowered (if slack exists). While BQL is well integrated into many wired Ethernet drivers [19], the `usbnet` driver has so far lacked this support.

F. Measuring Queuing-Induced Latency Under Load

Latency in communication networks is strongly dependent on the given network load [5], as queuing delay only manifests under sustained traffic and is absent under idle or lightly loaded conditions. To accurately characterize bufferbloat, measurements must be performed while bulk traffic keeps the queues continuously backlogged. The Flent test suite [20] with its Realtime Response Under Load (RRUL) workload is a widely used bufferbloat benchmark that saturates the link with bidirectional bulk flows while concurrently probing latency, typically using the standard `ping` tool in the background. For high-accuracy delay measurements, however, `ping` is a limiting factor, which motivates dedicated probing tools.

Precise delay measurement in 5G networks is challenging: Achieving sub-millisecond accuracy on a non-real-time OS without hardware timestamping or OS modifications requires careful software design. The `udp-ping` tool [21] addresses this by enforcing nearly constant inter-packet spacing to systematically uncover periodic effects that the standard `ping` tool would miss. It can run either on two time-synchronized hosts or, as in our setup (Sec. V), on a single host acting as both sender and receiver, which eliminates the need for external clock synchronization. It measures one-way delay rather than RTT, following the rationale of RFC 7679 [22], which advocates one-way metrics to avoid ambiguity from asymmetric paths and to prevent double-counting queuing delays that would otherwise appear in both directions of a round-trip. An additional advantage of one-way delay over ICMP ping in cellular setups is that ICMP may be prioritized or rate-limited by the Radio Access Network (RAN), introducing systematic bias. We adopt `udp-ping` in our evaluation.

III. Analysis of the usbnet Transmit Queue

The usbnet driver is the common framework for USB-attached network devices in the Linux kernel, covering both USB-to-Ethernet adapters and USB-connected cellular modems, as described in Sec. I. Many USB Ethernet adapters rely directly on usbnet, while others use device-specific drivers. Cellular modems typically use the qmi_wwan driver, which internally relies on usbnet for packet transmission. Before a packet is handed to the USB subsystem for transmission, it is placed in a software transmit queue inside the driver. This queue exists to ensure that packets are readily available whenever the USB host controller performs a new transfer, thereby keeping the link fully utilized.

The size of this transmit queue is set statically based on the USB generation:

- **USB 2:** The queue holds up to $60 \times 1518\,\text{B} \approx 91\,\text{KB}$.
- **USB 3:** The queue is five times larger at $\approx 455\,\text{KB}$.

While a large queue prevents transmitter starvation, it introduces a standing queue that adds latency. The worst-case queuing delay T experienced by a packet appended at the tail of a full queue of size Q is implicitly defined by the condition that all Q bytes ahead of it have been drained:

$$\sum_k r_k \cdot \Delta t_k = Q \tag{1}$$

where r_k is the link rate during the k-th transmission slot of duration Δt_k. Assuming a constant link rate R, this simplifies to:

$$t_{\text{queue}} = \frac{Q}{R} \tag{2}$$

where Q is the queue size in bytes and R is the link rate in bytes per second. As an example, for a USB 3 device operating at 10 Mbit/s, the queuing delay evaluates to:

$$t_{\text{queue}} = \frac{455\,\text{KB}}{10\,\text{Mbit/s} \;/\; 8\text{Bit/Byte}} \approx 364.3\,\text{ms}$$

For USB 2, the smaller queue of 91 KB still yields a delay of 72.86 ms at the same rate.

Importantly, the queue counts packets, and each packet in the usbnet queue is represented by an sk_buff structure that adds 232 bytes of kernel overhead beyond the network payload (measured using the default kernel). Consequently, small probe packets (e.g., the 50 Byte User Datagram Protocol (UDP) packets used in our measurements) effectively occupy $50 + 232 = 282$ Byte per slot, consuming a disproportionate share of the queue capacity. This overhead also means that the effective data payload per slot is smaller than the raw queue size, so real-world delays are marginally lower than the theoretical maximum.

This work addresses this by enabling BQL (Sec. II-E) in the usbnet driver, replacing the static queue limit with a dynamic one that keeps the queue just large enough to avoid starvation.

IV. Proposed Patch: BQL for usbnet

Prior to our patch, the usbnet driver used a static per-device transmit queue limit tx_qlen, fixed at 60 packets for USB 2 and 300 packets for USB 3, both independent of the actual link rate (see Sec. III). To replace this static sizing with BQL, the netdev_sent_queue and netdev_completed_queue functions described in Sec. II-E are wired into two code paths:

- **Transmit path (`usbnet_start_xmit`):** After a packet is successfully submitted to the USB subsystem, the netdev_sent_queue function is called to account for the bytes that have entered the device queue.
- **Completion path (`usbnet_bh`):** In the timer-driven completion handler, the netdev_completed_queue function is called with the number of completed bytes, allowing BQL to adapt the queue limit continuously.

In addition, netdev_reset_queue() is invoked on device open and stop to re-initialize the BQL state, and a dedicated spinlock prevents concurrent updates to the DQL counters. The patch has been accepted into the mainline Linux kernel [23], with two follow-up fixes addressing a corner case on device stop [24] and missing BQL accounting after resume [25].

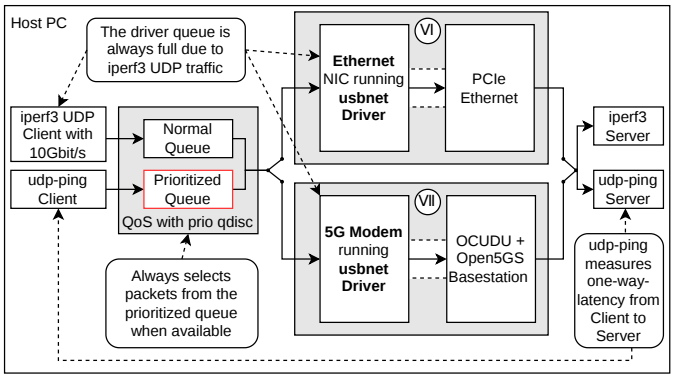

Fig. 2: Measurement setup for Ethernet and 5G cellular. A prio qdisc separates latency probe traffic from bulk UDP traffic. One-way delay is measured using udp-ping; time synchronization is inherent as sender and receiver run on the same host.

V. Measurement Setup

As illustrated in Fig. 1, multiple sources of latency exist along the transmission path. Network-internal delays (e.g., propagation and switching) are generally outside the user's control. To isolate the queuing delay introduced by the usbnet driver, we therefore design a setup that eliminates deep network paths, focuses on the host-to-first-hop segment, and avoids other bufferbloat mitigations.

Fig. 2 shows the Ethernet and 5G cellular configurations used in this work. In both cases, iperf3 generates bulk UDP traffic with a 2 MB socket buffer to ensure that the link is always fully utilized. Concurrent udp-ping [21] probes serve as a proxy for latency-sensitive application traffic and measure one-way delay under load (see Sec. II for the rationale behind this choice). The tool sends 50-byte UDP packets at 10 ms intervals from a 2 MB UDP send buffer, allowing precise characterisation of latency under sustained load and enough buffer space for the probe packets. To ensure that probe packets measure only the usbnet queuing delay and are not blocked behind bulk data, a prio qdisc is configured always to dequeue probe

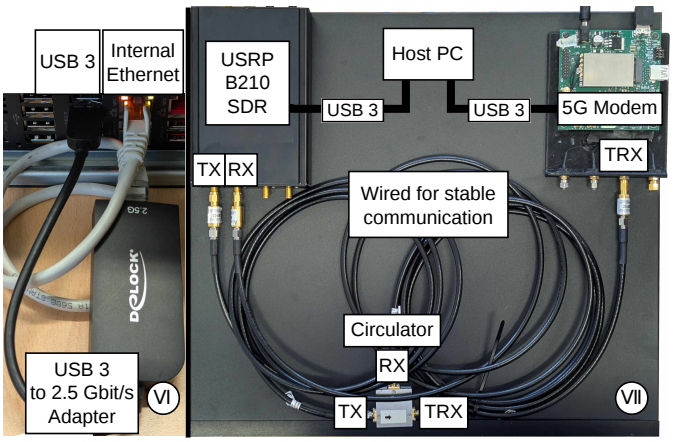

Fig. 3: Real-world hardware setup used for the evaluations. Left: USB-Ethernet dongle connected to internal LAN. Right: 5G setup with a USB modem cabled to an Software Defined Radio (SDR).

packets before `iperf3` traffic. Each measurement point was conducted over 300 s.

Fig. 3 provides photographs of the corresponding lab setups. For the Ethernet setup, the USB-to-Ethernet adapter under test is connected via USB 3 to the same host (AMD Ryzen 9 9950X), where the receiver uses the internal mainboard Ethernet NIC. For the 5G setup, a 5G USB modem is connected to the host via USB 3. The antenna ports are cabled to a USRP B210 SDR, on which an OCUDU gNodeB (gNB) [26] and an Open5GS core run, both on the same host. Because sender and receiver share the same host, the clocks are inherently synchronized, enabling accurate one-way delay measurements. The Linux kernel 6.18 is used for measurements without BQL; kernel 6.19, in which the patch was merged upstream, is used for the BQL-enabled measurements.

VI. Evaluation on USB-to-Ethernet Adapters

Using the Ethernet measurement setup described in Sec. V, evaluations were conducted with several USB-to-Ethernet adapters at different link speeds (see [23]). Here, we limit ourselves to the Delock 66045 USB 3 to 2.5 Gigabit Ethernet adapter (ASIX AX88279 chipset).

Fig. 4 shows side-by-side violin plots for each link speed. Six configurations are compared: fixed 1-packet queue, fixed limits of 100, 200, 300, and 455 KB (default, stock kernel), and the BQL-selected limit.

The data confirm that probe latency grows linearly with the queue limit, consistent with the model introduced in Eq. (2). BQL achieves substantial latency reductions while maintaining full throughput: at 100 Mbit/s, the prioritized probe one-way delay is reduced by a factor of **6.7**×, and at 2500 Mbit/s, by a factor of **7**×. At 100 and 1000 Mbit/s, even the 1-packet queue does not reduce throughput. By contrast, at 2500 Mbit/s the 1-packet queue lowers throughput to below ∼2000 Mbit/s, whereas BQL sustains the full line rate while still reducing latency. BQL is therefore the better choice, as it automatically finds a queue limit that avoids this throughput penalty.

VII. Evaluation on 5G Cellular Modems

To assess the impact of BQL on cellular interfaces, we use the 5G measurement setup from Sec. V with three commercial modems: the Quectel RM520 (Qualcomm X62 SoC), the Telit FN980M (Qualcomm X55 SoC), and the Fibocom FM160 (Qualcomm X62 SoC). The OCUDU gNB uses a 20 MHz channel, 30 kHz subcarrier spacing on band n78, and Modulation and Coding Scheme (MCS) indices 0, 13, and 27 (low, medium, and high data rates). The utilized Time Division Duplex (TDD) pattern has a 10-slot (5 ms) period: 6 downlink slots, 1 special slot (8 downlink symbols), and 3 uplink slots.

Fig. 5 shows the results as violin plots. Throughput is not plotted because full link utilization was achieved for all queue configurations; the attained data rate for each MCS value is noted at the left of each subplot. The same six queue configurations as in the Ethernet evaluation are compared: one-packet queue, 100, 200, 300, and 455 KB (default), and BQL.

As in the Ethernet case, latency grows approximately linearly with the queue limit: At MCS 0, the linear growth of latency with queue size saturates for larger limits, whereas at MCS 13 and 27 the trend is clearly visible. With the default 455 KB limit, the median one-way delay reaches several hundred milliseconds at MCS 13 and 27, and even exceeds 10 s at MCS 0 due to the very low data rate at that setting. BQL reduces the median one-way delay by a factor of **1.4**× at MCS 0 and by

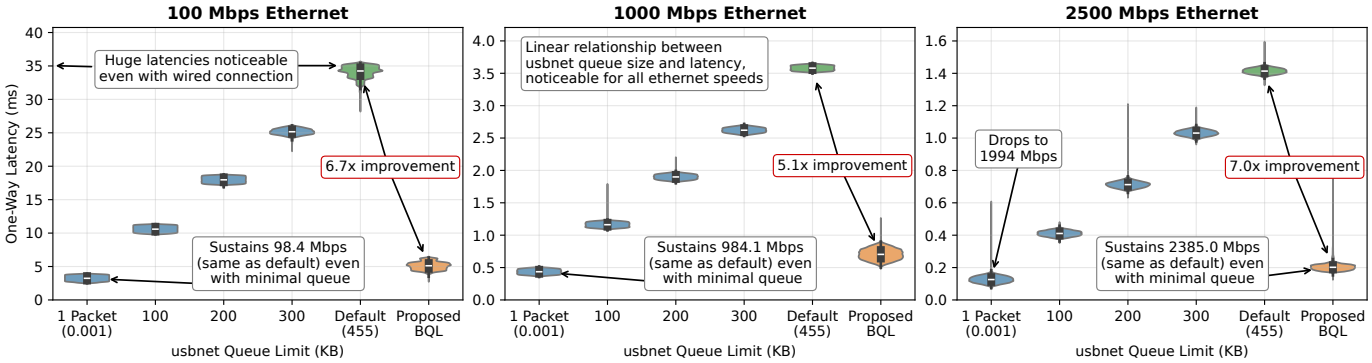

Fig. 4: One-way delay for the USB-to-Ethernet adapter. Latency scales linearly with queue limit. BQL reduces latency by up to 7× while sustaining full throughput; a 1-packet queue drops throughput below ∼2000 Mbit/s at 2.5 Gbit/s.

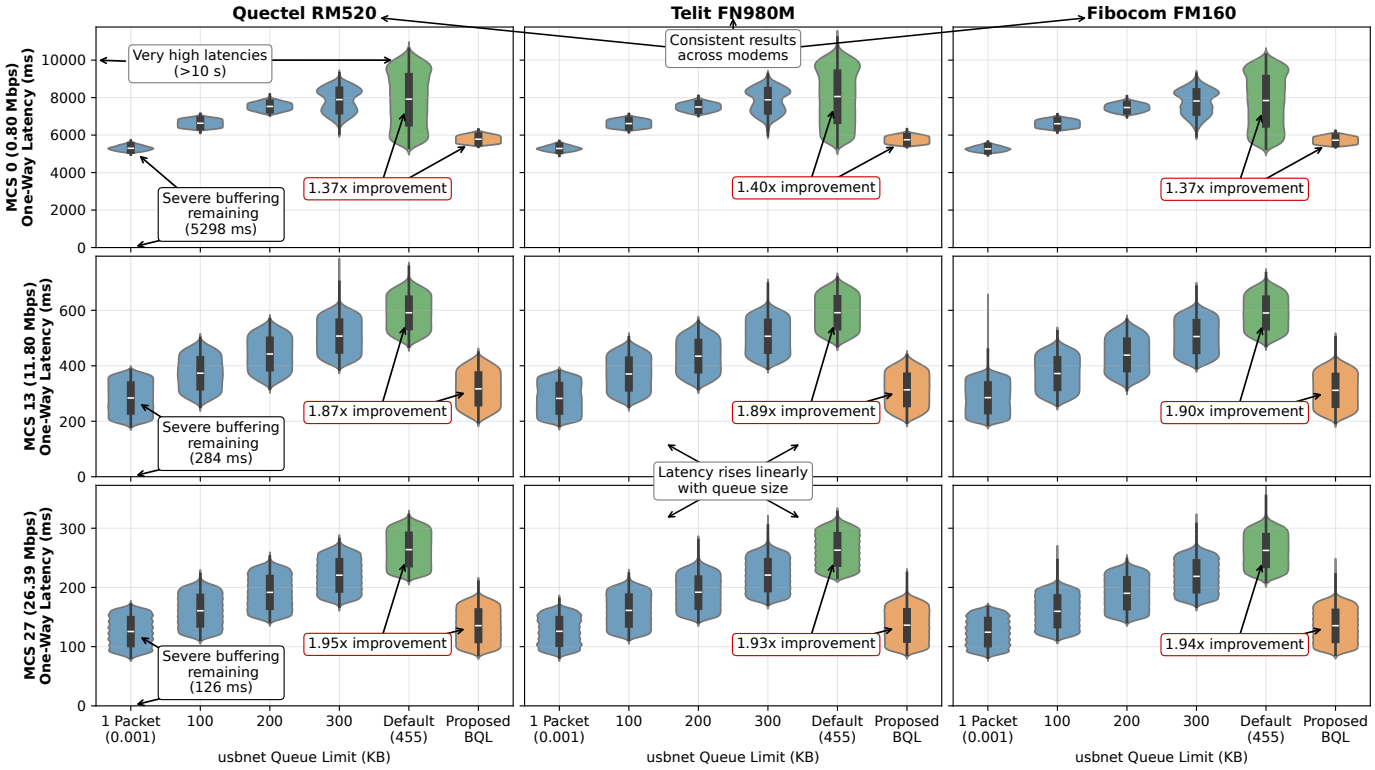

Fig. 5: One-way delay for the three 5G modems at MCS 0, 13, and 27. All modems show near-identical behavior, indicating System-on-Chip (SoC)-determined buffering. BQL reduces median latency by $1.9\times$ at MCS 13 and 27 and $1.4\times$ at MCS 0, while maintaining full throughput.

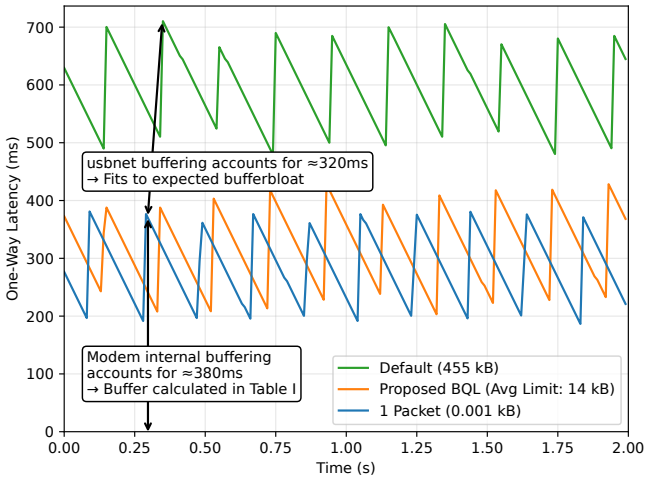

Fig. 6: Closer look at one-way delay time series for the Quectel RM520 modem at MCS 13. The sawtooth pattern reflects modem-internal buffering; BQL shifts it downward by $\sim$320 ms.

a factor of about **2**$\times$ at MCS 13 and 27. Notably, all three modems exhibit near-identical latency characteristics across all configurations, suggesting that the dominant buffering behavior is determined by the underlying 5G SoC rather than modem-specific firmware differences. These observations indicate that queuing inside the modem firmware contributes a baseline latency offset independent of the host-side `usbnet` buffer.

Fig. 6 shows the one-way delay time series for the Quectel RM520 at MCS 13. A periodic pattern is visible: the modem pulls packets from the host in bursts, in this case about every 200 ms, so delay rises as the queue fills and drops after each burst. This sawtooth behavior explains the high jitter in Fig. 5. With BQL enabled, the host-side queue is much smaller, shifting the entire pattern downward by about 320 ms while keeping a similar shape. Even with a queue limit of one packet, a residual median latency of around 280 ms (peaking at 380 ms) remains, attributed to modem-internal buffering unaffected by BQL.

To quantify this modem-internal buffering, we focus on the one-packet queue configuration, where the `usbnet` queue contributes negligibly, and the measured one-way delay reflects predominantly modem-internal delays. The maximum modem buffer size is estimated from the 99th-percentile latency as

$$Q_{\max} = \left(L_{99} - L_{\text{base}}\right) \cdot R, \tag{3}$$

where L_{99} is the 99th-percentile one-way delay, i.e., the upper 1% of latency samples that is intended to capture values near the top of the sawtooth, L_{base} is the unloaded baseline one-way delay, and R is the link throughput. Table I lists the results for the Quectel RM520, Telit FN980M, and Fibocom FM160. The unloaded baseline latency is similar across all MCS settings (about 13–19 ms). Despite the large variation in data rate across MCS values, the estimated modem buffer remains approximately constant at 480–560 KB for all three modems, suggesting a consistent internal buffer size, which is unchangeable to the knowledge of the authors.

TABLE I: Estimated modem-internal buffer size derived from one-way delay using Eq. (3).

Modem	MCS	L_{base} (ms)	L_{99} (ms)	R (Mbit/s)	$Q_{\max}$ (KB)
Quectel RM520	0	15.8	5614	0.8	560
	13	12.7	377	11.8	537
	27	13.2	166	26.3	502
Telit FN980M	0	13.8	5601	0.8	559
	13	13.4	375	11.9	537
	27	18.7	165	26.3	482
Fibocom FM160	0	13.1	5581	0.8	557
	13	12.7	378	11.8	539
	27	12.6	164	26.6	502

VIII. Conclusions and Outlook

This paper presented the integration of BQL into the Linux `usbnet` driver and evaluated its impact on both USB-to-Ethernet adapters and 5G cellular modems. An analysis of the existing static transmit queue revealed worst-case queuing delays of several hundred milliseconds, motivating the need for dynamic queue management.

For USB-to-Ethernet devices, BQL proved highly effective: the prioritized probe one-way delay was reduced by up to **7**$\times$ at 2.5 Gbit/s while maintaining full throughput across all tested link speeds. The measured latency scales linearly with the queue limit, confirming the analytical model.

For the three tested 5G cellular modems, the default 455 KB queue caused a median one-way delay of up to several hundred milliseconds at MCS 13 and 27, and above 10 s at MCS 0. BQL reduced the median one-way delay by about $2\times$ at MCS 13 and 27, and by $1.4\times$ at MCS 0. The latency–queue-size relationship was approximately linear, analogous to the Ethernet case. All three modems exhibited near-identical latency behavior, indicating that the buffering characteristics are shaped by the underlying 5G SoC rather than vendor-specific modem firmware. We calculated the modem-internal buffer to be approximately 500 KB. Our patch has been accepted into the mainline Linux kernel, benefiting billions of devices that rely on USB network interfaces.

In future work, the residual modem-internal buffering should be investigated in detail, and the feasibility of BQL-like dynamic sizing or user-configurable limits for the modem-internal buffer should be explored. The resulting latencies should be analyzed in teleoperation and industrial application contexts.

Acknowledgment

We thank the following people who helped with the Linux kernel integration:

- Eric Dumazet for the insightful feedback and bug fixing [24],
- Jakub Kicinski for applying the patch and maintaining the code [23],
- Bard Liao et al. for their bug reporting,
- Daniele Palmers for helpful tips regarding cellular modems.

This work has been funded by the German Federal Ministry of Research, Technology and Space (BMFTR) in the course of the 6GEM+ Transfer Hub under grant number 16KIS2412 and the VICTOR6G project under grant number 16KIS2549.

References

[1] F. Bouali, J. Pinola, V. Karyotis, B. Wissingh, M. Mitrou, P. Krishnan, and K. Moessner, "5G for Vehicular Use Cases: Analysis of Technical Requirements, Value Propositions and Outlook," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 73–96, 2021.

[2] 3GPP, "5G; Service requirements for enhanced V2X scenarios," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.186, Nov. 2020, Version 16.2.0.

[3] 3GPP, "Service requirements for cyber-physical control applications in vertical domains," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.104, 2024, Version 19.2.0.

[4] P. Popovski, J. J. Nielsen, et al., "Wireless Access for Ultra-Reliable Low-Latency Communication: Principles and Building Blocks," *IEEE Network*, vol. 32, no. 2, pp. 16–23, 2018.

[5] C. Arendt, S. Böcker, H. Schippers, T. Ploch, M. Kuhn, V. Venjakob, S. Hunger, and C. Wietfeld, "Towards Future Industrial Connectivity: Evaluation of Private 5G and Wi-Fi Networks in Professional Industrial Environments," in *2025 IEEE 21st International Conference on Factory Communication Systems (WFCS)*, 2025.

[6] J. Gettys, "Bufferbloat: Dark Buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, 2011.

[7] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," in *Proc. Int. Conf. Communications (ICC)*, vol. 3, 1979, pp. 43.1.1–43.1.10.

[8] K. K. Ramakrishnan, S. Floyd, and D. L. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168, Sep. 2001.

[9] E. Dumazet, *tcp: TCP Small Queues*, Linux kernel commit `46d3ceabd8d9`, https://github.com/torvalds/linux/commit/46d3ceabd8d9, Jul. 2012.

[10] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Commun. ACM*, vol. 60, no. 2, pp. 58–66, Jan. 2017.

[11] B. Briscoe, K. De Schepper, M. Bagnulo, and G. White, *Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture*, RFC 9330, Jan. 2023.

[12] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, *Controlled Delay Active Queue Management*, RFC 8289, Jan. 2018.

[13] R. Pan, P. Natarajan, F. Baker, and G. White, *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*, RFC 8033, 2017.

[14] E. Dumazet, *fq_codel: Fair Queue Codel AQM*, Linux kernel commit `4b549a2ef4be`, https://github.com/torvalds/linux/commit/4b549a2ef4be, May 2012.

[15] T. Høiland-Jørgensen, D. Täht, and J. Morton, "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways," in *Proc. IEEE Int. Symp. on Local and Metropolitan Area Networks (LANMAN)*, 2018, pp. 37–42.

[16] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom, "Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi," in *Proc. USENIX Annual Technical Conference (ATC)*, 2017, pp. 139–151.

[17] T. Gebauer, S. Schippers, and C. Wietfeld, "The NODROP Patch: Hardening Secure Networking for Real-time Teleoperation by Preventing Packet Drops in the Linux TUN Driver," in *2025 IEEE 102nd Vehicular Technology Conference (VTC2025-Fall)*, 2025.

[18] Z. Yi and P. J. Waskiewicz, "Enabling Linux Network Support of Hardware Multiqueue Devices," in *Proc. of the 2007 Linux Symposium*, 2007, pp. 305–310.

[19] T. Herbert, *Byte Queue Limits: The Unauthorized Biography*, https://medium.com/@tom_84912/byte-queue-limits-the-unauthorized-biography-61adc5730b83, Accessed: Apr. 2026, Jan. 2025.

[20] T. Høiland-Jørgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The FLExible Network Tester," in *Proc. 11th EAI Int. Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2017, pp. 120–125.

[21] M. Muehleisen and M. Abdel Latif, "Precise One-way Delay Measurement with Common Hardware and Software," in *Mobilkommunikation; 28. ITG-Fachtagung*, 2024, pp. 101–105.

[22] G. Almes, S. Kalidindi, M. J. Zekauskas, and A. Morton, *A One-Way Delay Metric for IP Performance Metrics (IPPM)*, RFC 7679, Jan. 2016.

[23] S. Schippers, *usbnet: Add Support for Byte Queue Limits (BQL)*, Linux kernel commit `7ff14c52049e`, https://github.com/torvalds/linux/commit/7ff14c52049e, Nov. 2025.

[24] E. Dumazet, *usbnet: Avoid a Possible Crash in dql_completed()*, Linux kernel commit `e34f0df3d81a`, https://github.com/torvalds/linux/commit/e34f0df3d81a, Dec. 2025.

[25] S. Schippers, *usbnet: Fix Crash Due to Missing BQL Accounting After Resume*, Linux kernel commit `c4efd7a770c5`, https://github.com/torvalds/linux/commit/c4efd7a770c5, Jan. 2026.

[26] Software Radio Systems, *OCUDU gNB: Open-source 5g base station*, https://ocudu.org/, Formerly srsRAN Project. Accessed: 2026-04-20, 2026.