Forewarned is Forearmed: A Responsive Congestion Control with Non-intrusive Uplink Dynamics Capture

Mu Wang[‡*], Yiying Lin[‡*], Shenghui Wei[‡], Enhuan Dong[§†], Kang Chen[§], Tong Li[⋆¶], Yinchao Zhang[§], Renjie Xie[•], Su Yao[§†], Ke Xu[§], Changqiao Xu[‡]

‡ Beijing University of Posts and Telecommunications, § Tsinghua University, ⋆ Renmin University of China, ¶ State Key Laboratory of Internet Architecture, • Guilin University of Electronic Technology

Abstract

The cellular uplink is increasingly important for modern applications but poses significant challenges to congestion control algorithms (CCAs), due to the highly dynamic conditions and complex operations. Existing CCA paradigms, mainly for downloading data, are unable to react promptly to uplink fluctuations because they rely on ACKs to indicate congestion, introducing inevitable control loop latency. Furthermore, the cost associated with modifications to both the sender and receiver cannot be overlooked.

We observe a fundamental opportunity in uplink scenarios: the sender's proximity to the wireless-side allows for fast and accurate control without relying on ACK-based feedback, and requires no modifications to either the receiver-side or the hardware layer. In this paper, we propose Aether, mainly consisting of: (1) a monitor deployed on the driver-layer that captures uplink dynamics, by leveraging our key finding that driver buffer dynamics serve as an indicator of uplink congestion; (2) a bottleneck detector that activates Aether when the uplink becomes the bottleneck; (3) a rate controller that steers transmission toward uplink capacity. We implement Aether on commercial devices and develop a trace-driven emulator for testing Aether. Extensive evaluations demonstrate that Aether outperforms state-of-the-art schemes, improving average throughput by 9.28% to 234.11% while consistently reducing latency by 5.34% to 56.94%.

CCS Concepts

• **Networks** → **Transport protocols**; **Mobile networks**.

Keywords

TCP congestion control, Cellular networks, Cellular uplinks, 5G/LTE, Modem driver information

ACM Reference Format:

Mu Wang, Yiying Lin, Shenghui Wei, Enhuan Dong, Kang Chen, Tong Li, Yinchao Zhang, Renjie Xie, Su Yao, Ke Xu, Changqiao Xu. 2026. Forewarned is Forearmed: A Responsive Congestion Control with Non-intrusive Uplink Dynamics Capture. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3789240.3829143

*Equal contribution.
†Corresponding author.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGCOMM '26, Denver, CO, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2467-1/2026/08
https://doi.org/10.1145/3789240.3829143

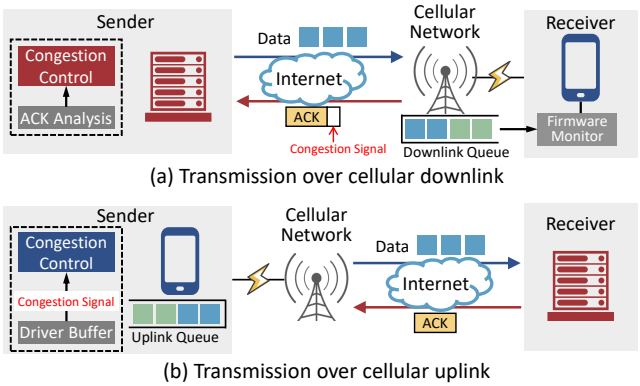


Figure 1: (a) Downlink transmission relies on ACKs to piggyback control signals from cellular infrastructure and UEs, introducing unavoidable control loop delay. (b) CCAs for uplink capture uplink congestion directly at the sender-side, enabling rapid reaction without feedback delay.

1 Introduction

The cellular uplink is increasingly important in the current Internet data transmission [30]. Emerging applications (e.g., crowdsourced livestreaming [47], video conference [36], cloud gaming [15], mobile AI inference [59]) necessitate continuous data upload from the client-side to the Cloud. When uploading data via cellular networks, it is essential to ensure the efficient utilization of uplink capacity, which often emerges as the bottleneck. Thus, mobile-side congestion control algorithms (CCAs) that govern the data upload rate become a pivotal determinant of uploading performance.

However, cellular infrastructure is *closed* and employs complex data scheduling [27, 29, 34, 44, 55]. Particularly in uplink scenarios, this opacity and complexity pose significant challenges to CCA design, necessitating rapid responses to network dynamics and precise rate control to safeguard transmission performance. Uplink transmission states, such as the modem status of user equipment (UE) [54], handover timing [43], and physical-layer resource scheduling [50], serve as critical indicators of wireless channel fluctuations. Acquiring and leveraging this information is key to realizing efficient cellular CCAs. The rationale is straightforward: the CCA's sending rate must closely align with the data scheduling behaviors of the cellular subsystems. This alignment ensures the full utilization of allocated wireless resources while preventing excessive data queuing that would otherwise inflate self-induced delay.

Fine-grained cellular information is important to ensure transmission performance, but the timely acquisition of such radio-originated information is non-trivial: ❶ Current ACK-based feedback designs are inefficient for uplink-bottlenecked transmission.

Figure 1 illustrates the structural differences between CCA control loops in cellular downlink and uplink scenarios. Existing cellular-oriented designs [20, 38, 39, 54] primarily target the server-side sender control and depend on UEs' ACKs to infer cellular network states. While downlink senders (i.e., servers) are compelled to rely on such feedback, this round-trip dependency is superfluous to the uplink. Since the UE acts as the sender, utilizing ACKs to echo cellular information back is fundamentally redundant. Instead, the sender can observe cellular variations directly, bypassing the feedback loop and avoiding observation delays; ❷ ACK-based feedback designs necessitate modifications at both endpoints (or even infrastructure-side), which are infeasible in production. Deploying such a design across distinct administrative domains—handset vendors, service providers, and network operators—is practically difficult; ❸ Real-time capture and analysis of cellular raw signaling messages in existing approaches [38, 54] suffer from substantial overhead and often require intrusive modem firmware interventions. *Thus, to ensure deployability, CCAs for the cellular uplink should be sender-centric, without requiring external coordination (e.g., receiver-side or infrastructure changes) or modem firmware modifications.*

We bridge the above mismatches by developing Aether, a responsive and lightweight CCA for cellular uplink. While the key challenge lies in capturing cellular network dynamics quickly and accurately without modifying the cellular subsystem or the receiver side, Aether addresses this through: ❶ Direct sender-side probing. Leveraging the sender's proximity to the air interface in uplink scenarios, Aether directly probes cellular transmission states at the UE, eliminating the unnecessary observation delay; ❷ Capturing cellular dynamics without modem firmware modifications. Our key findings (§3) reveal that the driver-level sending queue dynamics closely mirror the underlying modem data scheduling behaviors. Thus, Aether deploys the monitor on the driver layer to accurately capture the cellular dynamics (§4.3), avoiding modem hardware/firmware modifications; ❸ Bottleneck detector and rate adaptation. Aether applies a bottleneck detector (§4.4) that adaptively determines whether to invoke its rate controller based on the current bottleneck location. Furthermore, using the uplink capacity estimated by the packet driver-level dwell time (§4.5), Aether's rate controller quickly and precisely regulates the sending rate to adapt to network fluctuations while maintaining consistently low latency.

We implement Aether on Android smartphones and Linux laptops, with both integrated and PCIe-based modem chipsets (e.g., Qualcomm SDX55, Qualcomm SDX75, MediaTek Dimensity 1080). We develop a reproducible trace-driven emulator based on Mahimahi [45], extending it with a module that emulates driver queue dynamics. We collected real-world traces to capture how driver queues fluctuate with network dynamics and used them to drive the emulation. Both real-world and trace-driven results demonstrate that Aether achieves the highest throughput and the lowest round-trip time (RTT). Specifically, Aether outperformed representative CCAs such as CUBIC [21], Copa+ [26], Cesar [50], and Proteus [42], achieving throughput improvements up to 51.23%, 119.97%, 98.64%, 53.25%, along with RTT reductions of 28.14%, 20.6%, 20.31%, 28.53%, respectively (§6.2). Our key contributions are:

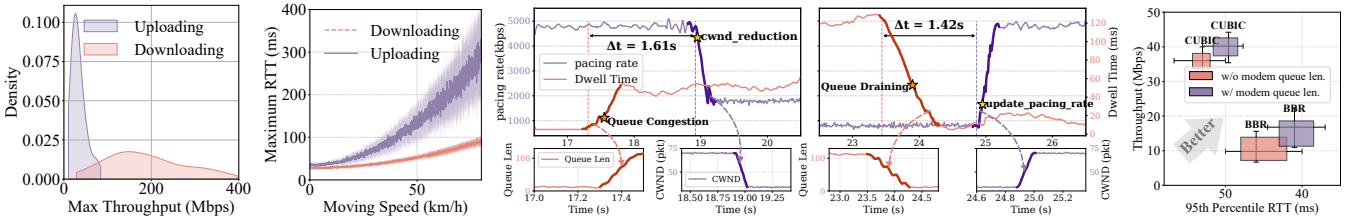
- We report, for the first time, a tight correlation between the modem hardware queuing behavior and its driver buffer state during cellular uplink transmissions, and demonstrate how this correlation can be leveraged to reliably and rapidly track uplink state without any intrusive modifications to cellular modem firmware or hardware.
- We propose a responsive CCA that uses the modem's driver buffer state for responsive, accurate rate adaptation over the cellular uplink.
- We deploy Aether and develop an emulation tool that replays traces with modem driver queue backlogs. We conduct extensive experiments in both real-world and trace-driven emulations. We released the code of Aether core implementations, the emulation tool, and collected anonymous traces[6].

2 Background

Transmission over cellular uplink. Default network operator configurations are downlink-favored [50], and the uplink capacity of LTE/5G is *much lower* than that of the downlink. We collected real-world cellular traffic traces from 61 base stations (BSs) owned by ChinaNet across a district of the Hangzhou urban area. The measurement spanned two weeks, with daily observations conducted between 10:00 and 19:00. The bands of the base stations include 800 MHz, 1.8 GHz, and 2.1 GHz. The maximum number of concurrent uplink connections per BS ranges from 45 to 1,405. We selected long-duration flows (>60s) corresponding to file transfers and video streams, as such flows are more likely to saturate the available uplink/downlink bandwidth. Moreover, the traces also include our maintained continuous iperf-based upload/download sessions to a local Cloud center (aiming to probe the maximum bandwidth). Figure 2a shows the average uplink and downlink data rate of the collected sessions. We can observe that the average uploading bandwidth of users is 16.7%-18.1% of the downloading bandwidth. Furthermore, the uploading bandwidth between different base stations is also highly varied, i.e., the bandwidth difference can be up to 28.31 Mbps for the neighboring stations. The uplink latency increases more rapidly than the downlink latency. Figure 2b shows that the delay when using the uplink increases more rapidly than that of the downlink. When the velocity reaches 80 km/h, the maximum uplink latency is over twice that of the downlink.

Limitations of ACK-based feedback. Current CCAs targeting cellular networks focus on data downloads, manifested by the assumption that the UE acts as the receiver. These methods rely on feedback loops where cellular subsystem metrics are piggybacked onto ACKs to drive the sender's rate controller. For instance, C2TCP [8] and Cesar [50] measure the cellular change via received ACKs, PBE-CC [54] embeds physical-layer bandwidth information into ACKs, while M2HO [38] utilizes ACKs to convey corrected window size. However, the fast variation in the cellular network renders these ACK-based feedback loops unable to keep pace with the actual network evolution. Cellular network states evolve at sub-RTT granularity. For example, physical resource allocation in BSs operates at the millisecond level [14] to track the rapid radio fluctuations. Consequently, the discrepancy between the sender's perception and the ground truth accumulates over time, triggering self-induced queue buildup and leading to congestion.

Challenges of modifications to both endpoints. Prior research [54] indicates that visibility into cellular radio side operations is



(a) Bandwidth Comparison (b) Delay comparison (c) Correlations between sending rate and modem metrics (d) Performance gain

Figure 2: (a) Uploading/Downloading rate difference: Downloading rate of most BSs is within [100,250] Mbps, some reaching 400 Mbps; Uploading rate is primarily concentrated in a narrow range [20,40] Mbps. (b) The RTT of uploading is more sensitive to UE speed: The uploading RTT can rise to about 319ms in average with speed increment, while the RTT of downloading remains largely below 100ms. (c) Case study of network varying with modem transmission status: when sudden congestion occurs (left), we first observe queue build-up at the modem. After a noticeable delay (1.6s), TCP's cwnd reduces. The queue requires time to drain the excessive injected data, rising self-induced delay. The plots below show that the modem-side queue length and cwnd exhibit correlated dynamics. When cellular recovers (right), we observe a sharp drop in packet dwell time over the modem, yet only after fully recovered the corresponding sending rate increase. (d) Performance gain of applying the queue length of the modem transmission buffer as a congestion signal (when ≥ 0.8x) to CUBIC, and BBR within the kernel.

indispensable for accurately perceiving cellular network states. However, schemes relying on explicit feedback [20, 54] often necessitate cross-entity collaboration, requiring the deployment of monitoring agents on UEs/BSs and modified CCAs on servers. Such designs mandate dual-ended modifications and infrastructure-level support, which face significant hurdles in production environments. In the real world, the network ecosystem is highly siloed: service providers update application logic, smartphone and chipset vendors control the UE's firmware/OS, and mobile operators operate the infrastructure. Since the sender and receiver are managed by distinct administrative domains, bridging these diverse stakeholders to implement a unified congestion control framework is operationally infeasible. Consequently, the reliance on multi-party participation renders these solutions largely theoretical, limiting their viability in large-scale deployment.

In fact, we argue that the reliance on ACK-based feedback is unnecessary in uplink scenarios. Since the sender in uplink scenarios is the UE, CCAs are able to access cellular states locally and instantaneously, eliminating the feedback latency inherent in network traversal. Leveraging this locality, a superior strategy is to monitor network fluctuations at the device side and directly expose them to the CCAs within the OS kernel. In the next section, we demonstrate how the modem driver buffer status can serve as an accurate, real-time signal for adjusting the sending rate.

3 Key Observations

Observation 1: Significant challenges in acquiring and analyzing cellular air interface operations. Current approaches [38, 54] mainly use signaling messages collected at the MAC/physical layer to measure the dynamics. However, acquiring such fine-grained signaling messages is non-trivial. Capturing cellular signaling messages relies on specialized diagnostic tools for interception and parsing. Such a diagnostic tool [34] can incur an extra CPU overhead, reaching 12%–23%. Crucially, the processing overhead is proportional to the message volume. When signaling messages surge during complex scenarios such as handovers (peaking at 20–30 messages per second), the computational burden is amplified. Most existing methods for real-time signaling collection necessitate intrusive modifications at the modem hardware/firmware level. In practice, the vast majority of smartphone vendors utilize proprietary baseband chipsets supplied by third-party manufacturers, making these methods difficult for deployment on commercial UEs.

Observation 2: Modem driver buffer status timely and accurately characterize cellular uplink status. To bypass the limitations of invasive modem firmware modifications and avoid the overhead of control signaling analysis, we scrutinize the data scheduling behavior within the modem subsystem. We collect real-world traces by leveraging vendor-supplied diagnostic tools and deploying hooks within the kernel TCP stack. These traces specify the packet states and processing dynamics at both the transport layer and the modem chipset. We select several traces and showcase the results in Figure 2c. With the degradation of the physical-layer transmission rate, the packet dwell time, which is termed as the time interval from packet entry into the modem buffer until the occupied buffer is freed, rises rapidly. Cellular uplink applies grant-based access (driven by Buffer Status Reports to the BS) and acknowledgment mechanisms to guarantee transmission reliability [37, 57]. Consequently, upon entering the cellular subsystem, packets remain buffered until acknowledged. Hence, network quality fluctuations are directly mirrored in modem transmission queue dynamics. The figure further reveals a *distinct lag* in TCP sending rate adjustments compared to the cellular physical-layer transmission rate. This latency arises because CCAs, such as BBR and CUBIC, are ACK-driven. The feedback from the receiver inherently delays the response to these network changes.

To demonstrate the efficiency of monitoring modem transmission queuing, we build a proof-of-concept test: We implement user-space BBR/CUBIC, and feed the modem dwell time to these CCAs. We modified the CCA's pacing rate update logic to use a weighted average (with a coefficient of 0.5) of two components: the standard pacing rate derived from ACKs, and the rate estimated by modem dwell time. As validated by real-world testing, Figure 2d demonstrates measurable improvements compared with the vanilla version of BBR and CUBIC. However, accessing the queuing status

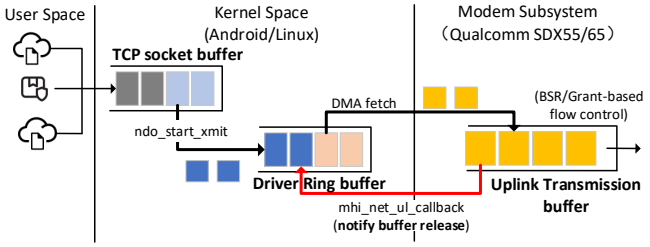
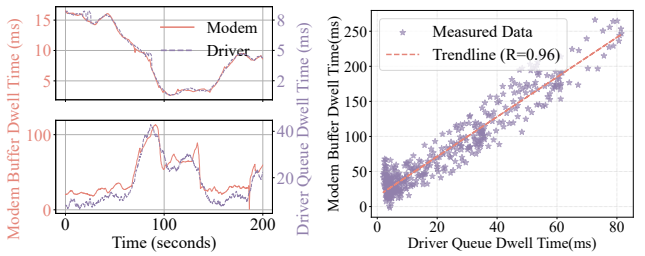


Figure 3: Interplay between modem driver and hardware, Qualcomm SDX55/65 as an example.



(a) Temporal queue dynamics **(b) Driver–modem correlation**

Figure 4: Correlation between the driver queuing and the modem transmission buffer.

of the modem transmission buffer still requires modem firmware-level modifications.

Observation 3: Correlation between driver queue and modem transmission buffer. We further investigate the hardware architectures and driver code of mainstream chips, such as those from Qualcomm, with the details summarized in Table 1 of Appendix A. Data interaction between the modem driver and the chipsets is facilitated by a DMA-based ring buffer. Taking the Qualcomm SDX55/65 chipset family as an example, as shown in Figure 3, packets delivered by the kernel stack are mapped to this shared ring buffer (TX ring) via the `ndo_start_xmit` function. Once the data is successfully enqueued into the modem's internal transmission queue, the driver releases the corresponding TX ring space using the function `mhi_net_ul_callback`.

Since the data transfer from the ring buffer to the modem's DRAM occurs via the PCIe/NoC interface, the transfer latency is negligible (microsecond level). Consequently, queue buildup at the driver level is primarily dictated by backpressure originating from the modem's on-chip transmission queue. Crucially, our trace analysis validates that when the uplink becomes the bottleneck, the packet residence time within the driver-layer buffer tightly couples with that inside the modem's internal transmission queue. Figure 4a indicates that when the wireless link becomes the transmission bottleneck, the queue buildup in the driver layer exhibits a high degree of consistency with the state of the modem's internal transmission buffer. Figure 4b further confirms a strong positive correlation between the two dwell times across all traces. More results are provided in Figure 16 in Appendix A. Under uplink saturation, the driver and modem transmission buffer effectively function as a single logical queue, resulting in consistent dwell time dynamics across both layers. A detailed formal proof is provided in Appendix A. Thus, packet driver-level dwell time can accurately infer the real-time cellular congestion level without requiring complex hardware/firmware changes.

4 Aether Design

In this section, we introduce the design insights and framework of Aether, and further elaborate on its detailed design.

4.1 Design Insight

Control driven by sender-side signals. High-performance CCAs for cellular uplinks require real-time visibility into uplink dynamics. Our key observations (§3) highlight the potential of leveraging the driver-layer buffer status at the UE's modem to achieve accurate and real-time capture of uplink dynamics. Compared to applying ACK-based feedback, monitoring the driver-layer buffering provides explicit signals that can immediately notify the sender at the beginning of wireless bottleneck queuing. Furthermore, tracking driver-layer operations does not require modifications to UE hardware/firmware, receiver-side, or network infrastructure.

Cellular uplink-aware rate control. Leveraging this driver-layer tracking, Aether uses the packet driver-level dwell time to measure the uplink condition changes and control the sending rate. The driver-level dwell time serves as an accurate and responsive proxy for the uplink congestion. It reflects the queue buildup at the uplink bottleneck, regardless of the underlying network environments or hardware configurations. Additionally, driver-level dwell time facilitates bottleneck localization without requiring network-side support. For example, by comparing the driver-level dwell time with RTT, the sender can distinguish whether congestion is stemming from the radio side or the Internet.

Non-goals and scope of Aether. Aether is designed specifically to boost cellular uplink transmission performance, rather than being a general-purpose CCA. This focus stems from the pronounced asymmetry between uplink and downlink characteristics. In practice, cellular radio resources are heavily skewed towards the downlink, leaving the uplink with "just enough" resources [18]. This makes the uplink far more sensitive to wireless dynamics as we discussed in §3, requiring additional, dedicated means to ensure rapid responsiveness. Downlink and uplink CCAs also diverge in design purpose. Server-side senders in downlink, with abundant resources, can afford computationally intensive algorithms (e.g., learning-based [17, 24, 42]) and are optimized for specific applications. Constrained by limited hardware resources and diverse application requirements, uplink CCAs must remain lightweight yet resilient. Notably, our uplink-centric design does not preclude downlink utility. We further elaborate on adapting our mechanism to downlink case in Appendix G.

4.2 Aether Overview

Figure 5 illustrates the architectural overview of Aether, which works as an add-on over commodity kernel CCAs (e.g., CUBIC and BBR). Aether consists of three blocks: ❶ transmission status monitor (§4.3), ❷ bottleneck detector (§4.4) and ❸ Aether rate controller (§4.5).

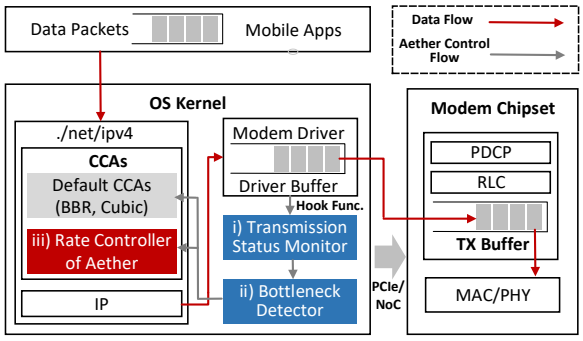


Figure 5: Overview of Aether.

Transmission status monitor. Guided by our first design insight, Aether introduces a transmission status monitor that captures packet-transmission events by deploying hooks at the driver level and measures the inter-packet transmission intervals.

Bottleneck detector. Guided by our second design insight, this module acts as a switch to determine whether Aether should intervene according to the modem driver buffer status, such as queue length of pending packets, and the transmission interval between consecutive packets (driver-level dwell time). Armed with this detector, Aether's control logic is activated exclusively when the radio access link constrains the uplink; otherwise, the detector immediately relinquishes control to standard kernel CCAs.

Rate controller. Once Aether's rate controller is activated, Aether temporarily calculates the optimal sending rate by driver-level dwell time. Aether gradually adjusts the `cwnd` value to approach the optimal rate in a step-wise manner (applying weighted window scaling policy), ensuring smooth variations in the sending rate. Furthermore, Aether is specifically designed to handle sudden uplink fluctuations and potential bottleneck shifts.

4.3 Transmission Status Monitor

Transmission status monitor addresses two key problems: ❶ how to monitor the driver-layer operations; ❷ which signals should be monitored to indicate uplink dynamics.

Driver-layer operations monitoring. To solve the first problem, we instrument the driver-level transmission queue (TX ring buffer) by placing hooks on its key manipulation routines (Enqueue and Dequeue function calls). To better illustrate the design, we use the open-source Linux kernel, which is also widely used by UEs (as the upstream OS kernel of Android), as an example. According to the device tree, the host interface of the modem mainly uses on-chip interconnects (e.g., Snapdragon 8 Elite, MediaTek Dimensity 1080) and external buses like PCIe (e.g., Snapdragon 865 with SDX55) or USB (Quectel chipsets). Distinct host interconnection interfaces correspond to specific driver modules within the Linux kernel: `mhi_net` for PCIe-based, `rmnet_ipa` for SoC-based, `qmi_wwan` for USB. UEs such as smartphones and laptops primarily adopt SoC-integrated or PCIe-based interconnections, driven by the need for high-speed data processing (see Appendix B).

Although these drivers differ in function organization and variable naming, their design logic for operating the TX ring is consistent. The analysis of driver source code shows that the data backlog dynamics within the TX ring are manipulated by packet transmit function calls (enqueue) and modem consumption signals (dequeue). In Appendix B, we showcase the related function calls in different drivers. As described in §3, in cellular networks, the driver's dequeue routine is tightly coupled with the modem's physical operation. Therefore, the invocation timing of this function is correlated with the uplink dynamics. Other mobile platforms, such as iOS and HarmonyOS, share consistent underlying cellular hardware architectures and bus protocols. Therefore, analogous transmission behaviors are observable across different platforms.

Indicators of cellular uplink dynamics. For the second problem, the metrics that vary with the air interface are the queue length and the driver-level dwell time (the time from when an individual packet enters the queue until it is removed). The maximum length of the `skb_buffer` differs by modem type and driver version (e.g., 4 for USB, 128 for PCIe). Therefore, if queue length were used as the sole CCAs parameter (i.e., using a queue-length threshold to decide whether to signal uplink congestion), each modem would require a customized algorithm to accommodate its specific maximum driver queue length. Instead, we use driver-level dwell time. This indicator can be regarded as the packet's sub-`rtt` on the air interface; using this sub-`rtt` makes it easier to estimate the current air-interface capacity and determine whether the air interface is the bottleneck. In addition, the gradient of this residence time can be used as a key indicator to report whether the network status of the air interface is changing.

4.4 Bottleneck Detector

Cellular network conditions fluctuate rapidly, which inevitably causes the bottleneck to shift from the wireless side to the non-wireless side during transmission. Aether detects the bottleneck location based on the observed driver-level dwell time and packet-level end-to-end latency.

Bottleneck shifting. We define two types of bottleneck: *uplink bottleneck*, where the cellular uplink is the bottleneck of the connection, *Internet bottleneck* for otherwise. When the TCP sending rate exceeds the modem's output rate, the uplink becomes the transmission bottleneck. When the bottleneck shifts to uplink, the driver queue builds up quickly and driver-level dwell time increases continuously. Our analysis shows that the driver queuing up arises primarily during startup or abrupt changes in network conditions, for example: (1) When a UE begins to upload data, UE sends a Buffer Status Report (BSR) and awaits a scheduling grant from the BSs prior to actual data transmission; (2) When the allocable Physical Resource Blocks (PRBs) decrease because the number of users served by the BS has increased; (3) When a packet needs to retransmit at the cellular MAC layer due to receiving of negative acknowledgment. Conversely, when the bottleneck shifts to *Internet bottleneck*, the driver's transmission buffer drains rapidly and driver-level dwell time falls to very small values.

Dealing with different bottlenecks. When *uplink bottleneck* is detected, Aether directly activates its uplink-dedicated rate controller (§4.5). When in *Internet bottleneck*, the uplink capacity may exceed the actual bottleneck bandwidth. Determining the sending rate based on uplink capacity may increase self-induced latency. To avoid this, the bottleneck detector immediately switches control to the default CCA.

Bottleneck detecting. Identifying whether the cellular network acts as the primary bottleneck is challenging, particularly given the lack of visibility into MAC-layer scheduling patterns [11]. We record driver-level dwell time D_k for each enqueued packet k, and sample the end-to-end $\texttt{rtt}_{e2e}$ from the tcp `rtt estimator`. If the D_k spikes yet the delay of the non-wireless link remains static, the current network is in *uplink bottleneck*. The non-wireless link delay can be approximated by $\texttt{rtt}_{non-k} = \texttt{rtt}_{e2e} - 2D_k$. The rationale for using double D_k is that the driver and buffer dynamics are correlated. While this metric serves as a proxy rather than a precise measurement of wireless latency, it provides adequate resolution for detecting link bottlenecks. Let driver-level dwell time difference in consecutive packets be denoted by $\Delta D_k : D_k - D_{k-1}$. Thus, when we observe ΔD_k increase rapidly and $\texttt{rtt}_{non-k}$ remains static, or the magnitude of the increase is less than ΔD_k, i.e., $\Delta D_k > \Delta D_{k-1}$ and $\texttt{rtt}_{non-k} \approx \texttt{rtt}_{non-k-1}$. Otherwise, the current network is in *Internet bottleneck*. Our evaluation (§6.4) confirms the effectiveness of using driver-level dwell time to estimate bottleneck conditions.

When UEs initialize a data uploading session, the bottleneck detector enters the *uplink bottleneck* by default. This design has two advantages: ❶ for most of the time, cellular uplink is the bottleneck of the established connections, as we discussed in §2; ❷ Entering *uplink bottleneck* triggers Aether control logic. Aether's rate controller (§4.5) can quickly converge to the actual uplink bandwidth, avoiding the startup data surges that would otherwise build a long transmission queue.

4.5 Rate Controller of Aether

The rate controller of Aether mainly consists of two components: ❶ uplink capacity estimator, ❷ rate controller.

Uplink capacity estimator. We consider two different cases of estimating uplink capacity: when the transmission queue is empty and when it is non-empty.

Non-empty queue. For the case that the driver transmission buffer is not empty (e.g., already in *uplink bottleneck* state, other apps or threads are delivering data), a stable value of driver-level dwell time can be observed. Guided by Little's law, the uplink capacity is characteristically modeled as:

$$C_u = \frac{1}{\gamma} \sum_{k=1}^{\gamma} \frac{Q_k}{D_k} \tag{1}$$

where Q_k and D_k are respectively, the queue length and the driver-level dwell time observed when k-th packet arrived, γ is the number of sampled packets.

Empty queue. When the driver's buffer is empty, Aether adopts a *probe-and-estimate* strategy (i.e., the UEs begin to upload data) to calculate the uplink capacity. Aether first sends K consecutive packets to probe the uplink capacity. K is a positive value smaller than half of the TX buffer size, which ensures that injected packets do not build up a long queue in a short time, avoiding prolonged driver-level dwell time and the self-induced latency enlargement. After sending K packets, Aether uses Equation 1 to calculate the capacity.

Notably, if no significant queue buildup is observed, it indicates that the current injection rate K has not yet reached the achievable uplink capacity. In this case, if the bottleneck detector confirms that the uplink remains the bottleneck, the next probing cycle doubles the K if the uplink is no longer identified as the bottleneck, Aether terminates and reports the current K to the default CCAs, allowing them to skip the initial states. The uplink capacity estimation process is shown in lines 5-21 in **Algorithm 1** (Appendix C).

Rate controller. We update the `cwnd` in a multiplicative manner. The updating formula should meet: ❶ the upper layer sending rate r should remain static when r equals C_u; ❷ the magnitude of changes (increases/decreases) in the sending rate should diminish as it approaches C_u. These two conditions ensure the sending rate approaches the uplink capacity as smoothly as possible and does not exceed it. Inspired by prior work [10, 20, 22], we set the multiplicative operator as a log function:

$$f(r, C_u) := 1 + \log \frac{C_u}{r}, r > 0 \tag{2}$$

When r equals C_u, the multiplier is 1, satisfying condition ❶. The log function is concave, and thus meets ❷. To further reduce latency and preserve fairness, we introduce a factor:

$$v(r) := \alpha D \times \min\{0, C_u - r\} \tag{3}$$

This factor becomes active when the sending rate r exceeds C_u. Since the queue length can be expressed as the product of the arrival rate and the waiting time, v can be physically interpreted as the contribution of the data flow to the growth of the driver transmission buffer. D is the average driver-level dwell time. α is a pre-defined constant, whose value impacts the data flow competitiveness. A smaller α indicates the flow is not sensitive to the buffer growth, which leads the data flow to favor a more aggressive sending strategy. From an optimization perspective, such a design can also be considered as a penalty for long queuing delay, as in Copa [10]. By Equation 2 and 3, Aether updates the `cwnd` by:

$$\texttt{cwnd}_t = \texttt{cwnd}_{t-1} \times f(p_{t-1}, C_u) + v(p_{t-1}) \tag{4}$$

where p_t is the pacing rate at t. Thus, the pacing rate is equal to $\texttt{cwnd}_t/\texttt{rtt}_{e2e}$, which is the same as CUBIC and has already been proved to be robust. Lines 22-27 in **Algorithm 1** (Appendix C) show how Aether controls the data rate by Equation 4. We provide a theoretical analysis of Aether's stability and fairness in Appendix D.

Resilience to bottleneck misdetection. In the rare event that Aether's rate controller is triggered when the bottleneck detector misjudges the state as the uplink bottleneck when it is not, the resulting short driver-level dwell time can lead to bandwidth overestimation. When consecutive events reveal that $\texttt{rtt}_{e2e}$ growth outpaces the increment of driver-level dwell time, indicating external network congestion, Aether preserves the robustness of the rate control loop by seamlessly switching to native kernel CCA. Specifically, it freezes `cwnd` updates, and invokes the control logic of native kernel CCA for upcoming window adjustment period.

Coping with sudden uplink degradation. Cellular networks can fluctuate frequently and drastically. For instance, available bandwidth can plummet from tens of Mbps to a few kbps instantly due to handovers during high-speed mobility or channel fading. Stepwise rate updates often struggle to keep pace with such drastic variations. Thus, Aether explicitly handles sudden uplink bandwidth drops using driver-level dwell time difference.

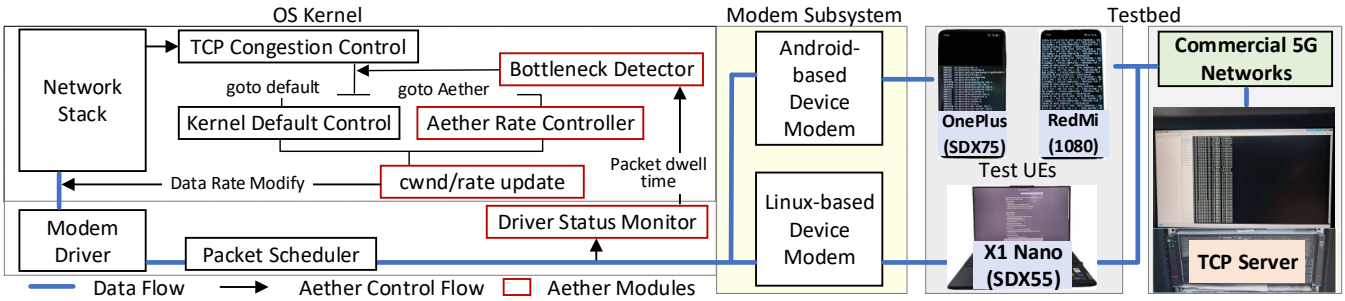


Figure 6: Aether Implementation and Testbed Overview.

When $\Delta D_k \geq C\Delta D_{k-1}$ is observed, we define that a sudden uplink bandwidth drop has occurred. In practice, $C \in [2.8, 4.5]$. Aether begins to drain the buffer and freezes window updates (see the lines 7-8 in **Algorithm 2** of Appendix C). When the driver buffer has drained, Aether records the draining time T. The current draining rate is then computed as Q_t/T, where Q_t denotes the data backlog in the buffer at the time of draining. Aether determines how to adjust the sending rate according to the draining time T: ❶ If $Q_t/T \geq C_u^-$, where C_u^- is the estimated uplink capacity before draining, Aether restores the cwnd and pacing rate to the values before entering the draining phase; ❷ Otherwise, Aether uses the draining rate as the data sending rate until the driver-level dwell time begins to decrease.

4.6 Design rationales

In the following, we discuss several key design choices of Aether.

Window updating. Regarding the design of the multiplicative factor, instead of directly adopting the ratio C_u/r, we integrate a logarithmic form. This design choice is motivated by the following considerations: C_u/r spikes sharply during the startup phase, leading to severe cwnd oscillations, which defeats the purpose of smooth pacing when designing CCAs. The logarithmic design dampens these volatile spikes, fostering a more moderate transition while preserving robust convergence when approaching C_u.

Bottleneck detection methods. To maintain the non-intrusiveness into the cellular subsystem, Aether cannot directly deploy physical-layer-based methods, such as BurstTracker [11], to locate the bottleneck. An alternative is the dual-control-loop design, where the default congestion control and Aether independently compute window sizes and select the minimum as the definitive cwnd. This approach can miss critical cwnd-growth opportunities. We instead directly measure the driver-layer queue dwell time against the total rtt. When the cellular link becomes the bottleneck, the variance of the driver-layer queuing delay will dominate the rtt fluctuations, providing a robust and distinct signature for localized bottleneck identification.

Interaction with Default CCAs. Upon detecting the *Internet bottleneck*, control is handed back to the native CCAs, requiring a well-defined interaction between Aether and other CCAs. Aether enables native CCAs to smoothly recover from its operating state. Since Aether tracks wireless capacity (which is currently non-bottlenecked), thus, the actual end-to-end throughput may be lower than Aether's estimation, the native CCAs probe the new available bandwidth. Taking CUBIC and BBR as examples, upon identifying the Internet as the bottleneck, CUBIC directly inputs the cwnd computed by Aether as the last maximum window size into its CUBIC function for window updating. BBR enters the ProbeBW phase to detect new bandwidth values. Other CCAs can similarly recover their respective cwnd from Aether.

5 Implementation

Aether confines all modifications strictly to the kernel space, requiring no modifications to the UEs' hardware/firmware, BSs, receiver, or applications. The implementation overview is shown in Figure 6. We present the deployment of Aether on Linux kernel-based operating systems (including Android, the dominant mobile platform). The implementation consists of three major components:

Monitor. The core driver interacting with the modem is `rmnet_ipa.c`, which adheres to the Linux Net_device_ops structure and defines device behaviors. The enqueue operation of driver TX buffer is handled by `ipa3_wwan_xmit` (corresponding to the Linux interface `ndo_start_xmit`). The dequeue operation is `apps_ipa_tx_complete_notify`. This function is called when the hardware releases the TX buffer, signifying the completion of hardware processing. To measure the driver-level dwell time, Aether hooks these function calls and logs invocation timestamps alongside packet metadata. We import the driver-level dwell time and buffer length into the kernel global symbol table via EXPORT_SYMBOL, allowing direct access by the kernel modules bottleneck detector and rate controller of Aether.

Bottleneck detector and rate controller. The Bottleneck Detector is deployed in the form of a kernel module. Bottleneck detector reads driver-level dwell time and buffer length from EXPORT_SYMBOL to determine whether the link is currently a bottleneck according to the logic in §4.4. The rate controller modifies the bottleneck signal bit accordingly. For rate controller's implementation, we directly add Aether's code (Algorithm 1) to the existing kernel CCAs. Taking CUBIC as an example, we embed the cwnd updates of Aether into the rate control parts in cubic.c, i.e., bictcp_update. We use the bottleneck signal bit as the condition for entering this logic: when the bottleneck signal bit is 1, the system enters Aether's rate controlling logic instead of adopting CUBIC's original window adjustment method.

We currently implement Aether mainly on Android phones with Qualcomm chipsets (integrated modem), Linux laptops (PCIe-based modem), and Android phones with MediaTek chipsets. The kernel

versions of tested devices are 6.5.13 (X1 Nano), 6.6.30 (OnePlus), and 4.19.191 (RedMi). Note that the relevant driver code can serve as a generic reference across other kernel versions. Since our design does not depend on support from underlying hardware, BSs, receiver, or applications, implementing Aether poses no technical barriers for smartphone vendors (including those dependent on third-party hardware suppliers) and can be readily deployed on existing commercial devices. For example, in the Android ecosystem [7] with the generic kernel image (GKI) architecture, cellular modem drivers are decoupled and deployed as independent vendor modules. Modifications confined to the internal logic of a vendor module do not violate the kernel module interface (KMI) stability [4]. Thus, it only necessitates recompiling the specific driver module (e.g., `rmnet_ipa`) and the CCAs module (net_ipv4_tcp_ccname), which reduces the engineering complexity for deployment. In Appendix G, we elaborate on the compatibility of Aether, confirming that the method is forward-compatible with future kernel and driver updates. We are currently collaborating with major domestic smartphone manufacturers to advance the deployment of this approach on commercial devices.

6 Evaluation

We conduct both real-world and trace-driven evaluations to validate Aether's performance. In general, we mainly evaluate Aether by the following aspects: ❶ We verify that Aether provides high throughput and low latency across a wide range of diverse real-world network conditions (§6.2); ❷ We test Aether in the trace-driven environment and assess its average throughput, delay, and ability to adapt to network changes (§6.3); ❸ We test Aether fairness, adaptation to bottleneck shifting and overhead (§6.4); ❹ We analyze the impact of different parameter settings (§6.4).

6.1 Experimental Setup

Real-world testbed. We build the testbed using commercial 5G networks. The testbed consists of client-side senders and a server-side receiver. Aether and baselines are deployed on several UEs, including OnePlus (Qualcomm SDX75), Redmi (MediaTek 1080), and ThinkPad X1 Nano (Qualcomm SDX55). These devices access the server via cellular links from China Unicom, Mobile, and Telecom. The server connects to the Internet with a 1 Gbps link to prevent server-side bottlenecks.

Baselines. For real-world tests, we demonstrate the advantages of Aether through comparative experiments with state-of-the-art CCAs, which are deployable within the Linux kernel. Baselines include loss-based methods such as CUBIC [21], delay-based methods such as various versions of BBR [13], and Copa+ [26], learning-based approaches like Vivace [17], Aurora [24], Proteus [42], Astraea [35], and cellular-oriented methods such as C2TCP [8] and Cesar [50]. Approaches that remain in the proof-of-concept (PoC) stage are evaluated exclusively in our trace-driven emulations.

Settings. Clients generate traffic to a server using iperf3 over the networks of three operators. Considering that the network conditions and resource scheduling policy can be different among the operators, at least 100 sets of valid data are collected for each condition. After removing outliers, the arithmetic mean is calculated as the performance indicator. We set the default Aether 's parameters to $\gamma = 5$, $\alpha = 0.6$, and $K = 12$ based on empirical results.

6.2 Real-world Delay and Throughput

Different signal conditions. To simulate various network conditions users may encounter in daily usage, we select three representative signal strength scenarios: ❶ Outdoors scenarios, featuring stable network connectivity with minimal interference, and consisting of limited movement (speed range [10,40] km/h). These scenarios are the primary use case for cellular networks; ❷ Indoor scenarios, with noticeable signal attenuation and moderate link quality. UEs in this scenario are stationary or moving at low speed [0,5] km/h. ❸ Underground parking lots, significant signal degradation due to poor radio conditions. We test the performance using three carriers over the outdoor scenario. The reason for choosing outdoors is its favorable radio conditions, which better reveal operator-specific effects from spectrum allocation, scheduling, and BSs placement. All tests are conducted at the same geographical locations and use the average obtained from repeated experiments.

Figure 7 demonstrates CCAs' performance under three scenarios. Aether showcases significant performance gains over mainstream congestion control algorithms. Specifically, compared to loss-based (e.g., CUBIC), delay-based (e.g., Copa+), cellular-specific (e.g., C2TCP), and learning-based (e.g., Aurora) algorithms, Aether increases average throughput by 51.23%, 119.97%, 24.6%, and 11.8%, respectively. Concurrently, it reduces the 95th percentile RTT by 28.14%, 20.6%, 48.61%, and 50.9%, respectively.

As the default CCA in the Linux kernel, CUBIC performs well across various network environments. Delay-based solutions, such as Copa+, show divergent performance and are often unable to achieve high throughput. Besides, the RTT stability profiles differ from one another and fluctuate based on the specific network conditions. BBR exhibits low link utilization in scenarios with strong signals. This may stem from flaws in its mechanism, where the sender estimates bandwidth based on the receiving rate and actively drains the queue to obtain $\mathrm{RTT}_{\min}$ [23], resulting in an inability to accurately estimate bandwidth-delay product (BDP). Both Vivace and Proteus, applying online learning, select the utility-optimal transmission rate through rate probing and perform well under various scenarios. However, Vivace has a higher RTT, while Proteus effectively mitigates the impact of RTT by incorporating delay variation into its utility function. Aurora achieves high throughput with the purpose to learn a better throughput, but it also leads to a significant increase in RTT. Astraea exhibits significant limitations across multiple scenarios. Learning-based algorithm performs exceptionally well only within well-trained cases [51]. Its performance declines markedly when deployed in unseen environments.

C2TCP is able to achieve high throughput across different scenarios, i.e., when in the outdoor case. However, the delay of C2TCP varies significantly among different scenarios. The degraded throughput of Cesar in uplink scenarios stems from its design, which mainly relies on the unique scheduling behaviors of downlink traffic. The algorithm is unable to accurately perceive the network state. Aether achieves both good throughput and low latency compared to other CCAs. Especially when in outdoor scenarios, Aether is the best among all tested CCAs. Benefiting from the design of Aether in monitoring the driver-level dwell time on the modem driver, Aether

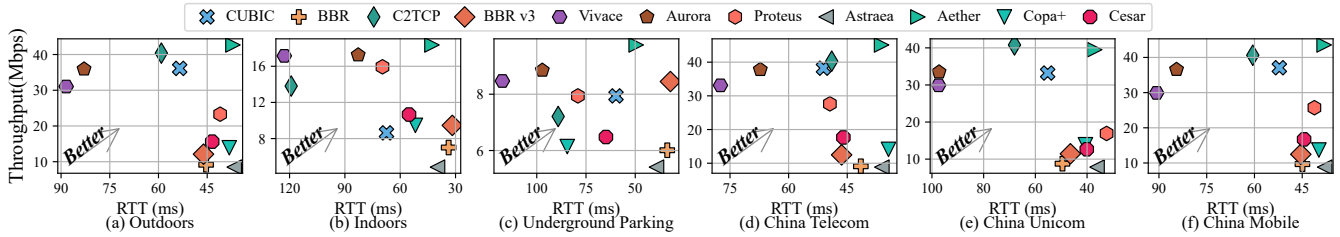


Figure 7: Performance comparison for real-world experiments.

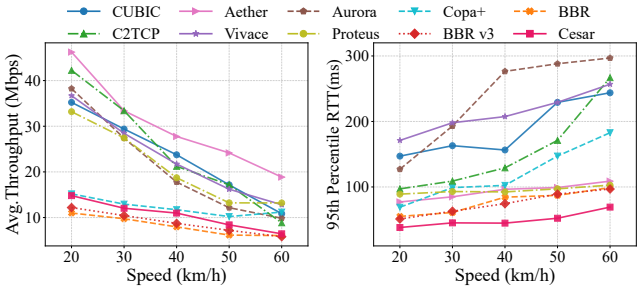


Figure 8: Mobility performance of CCAs.

can timely capture subtle fluctuations in uplink, enabling precise control of transmission rates and cwnd. Consequently, it ensures exceptionally high link utilization while effectively maintaining RTT at a consistently low level.

Mobility performance. Considering the network conditions when people use mobile devices in daily life, we conduct systematic tests in various typical public transportation scenarios. These test scenarios include mobile environments at different speeds, such as city buses (20–40 km/h) and taxis (40–60 km/h), covering users' common commuting situations. In the experimental design, we ensure a consistent movement path and specifically focus on several key characteristics of mobile network environments: frequent base station handovers, signal strength variations, and network latency fluctuations. By performing 100 tests on different CCAs, we use the average values of bandwidth and latency to represent their performance. Empirical evaluation under diverse signal conditions demonstrated that Astraea's performance in dynamic networks is unsatisfactory, thus warranting its exclusion from the present experimental study.

Figure 8 illustrates how the CCAs' performance varies with movement speed. We observe that Aether demonstrates a clear advantage in the critical metric of maximum link utilization. Specifically, across five different speed scenarios, Aether consistently maintains high throughput, indicating its strong adaptability to changing network conditions and efficient utilization of available bandwidth. Regarding RTT, a key indicator of network responsiveness, Aether performs slightly behind BBRv3. This result suggests that while delivering high throughput, Aether also maintains low transmission latency, achieving an effective balance between network efficiency and real-time performance. As indicated by Equation 4, Aether's control logic does not solely pursue minimal latency; instead, it carefully balances maximizing wireless bandwidth utilization against mitigating excessive queuing delays. Hence, Aether outperforms alternative methods in terms of throughput while maintaining a flat latency growth curve as the movement speed scales up.

6.3 Trace-driven Emulation

To benchmark against existing PoC approaches and assess system generalizability across different countries and regions, we also conduct trace-driven evaluations. Our emulation tools and complete trace dataset are available [6].

Emulator implementation. Current studies mainly use record-and-replay tools such as Mahimahi [45]. More recent CellReplay [49] improves accuracy in cellular networks via traffic interpolation between light and heavy workloads. However, these tools do not capture driver-level queuing operations and are unable to evaluate Aether, whose performance relies on the explicit driver-layer information. To address this limitation, we extend Mahimahi by adding a new component `virtual send queue`. This component simulates the residence time of packets in the driver layer and dynamic changes in queue length. We provide the detailed design and validation of the modified tool in Appendix F.1.

Dataset. We use publicly available cellular uplink traces spanning several geographically distinct regions: five US urban areas in North America [19, 49] and Belgium in Europe [52]. The North American traces span five major US metropolitan areas (Boston, Chicago, Indianapolis, Champaign-Urbana, and New York), covering Tier-1 commercial carriers (Verizon, T-Mobile, and AT&T) under hybrid 5G/LTE deployments. The European traces are collected in Belgium under mobile networks, covering three mobility scenarios: Walking, Cycling, and Driving. In addition to the North American and European traces, we also integrate over 30 traces collected from Beijing. Detailed descriptions are provided in Appendix F.2.

Baseline. In addition to the baselines used in §6.2, we include PBE-CC [54], a PoC cellular CCA using physical layer feedback. Direct comparisons with PBE-CC in prior literature are rare since fully implementing its design is difficult. Instead, we implement an idealized version of PBE-CC capturing its core design, skipping the complex hardware-based acquisition mechanisms and directly injecting ground truth bandwidth to the rate adaptation formula [54]. This idealized realization, free from measurement overheads, establishes the achievable upper bound by the algorithm.

Overall Performance. Figure 9 illustrates the throughput-delay trade-off across all emulation scenarios. Aether consistently achieves a good balance between high throughput and low latency. For example, in US traces, Aether performs within 4% of the idealized PBE-CC's throughput, effectively matching the theoretical upper bound while maintaining comparable latency. Also, in the challenging high mobility cases (i.e., US and Belgium Driving), Aether exhibits exceptional robustness. While conventional schemes such as CUBIC and

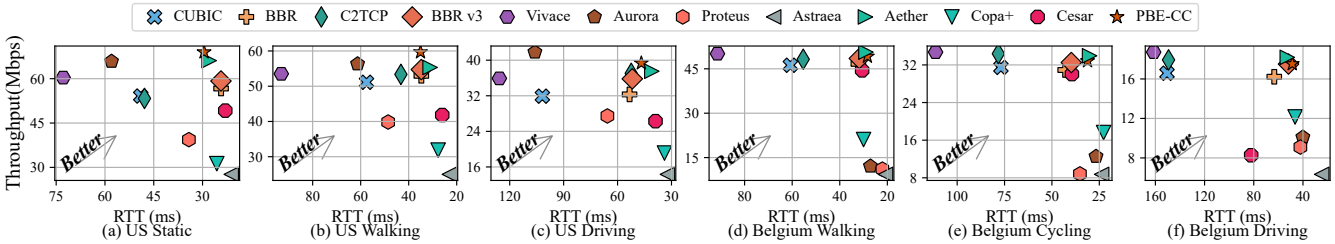


Figure 9: Performance comparison for trace-driven emulations.

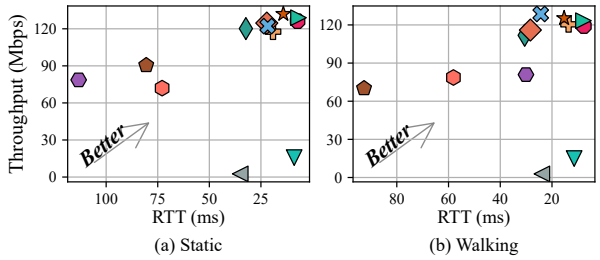


Figure 10: Overall performance in high bandwidth cases

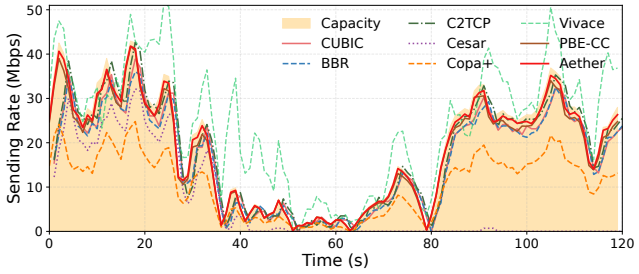


Figure 11: Adaptation to Bandwidth Fluctuations.

BBRv3 degrade by 20%-40% due to their inability to distinguish non-congestion signal fluctuations, Aether maintains high performance close to PBE-CC. The effectiveness of Aether is also confirmed on the Belgium traces. Aether maintains high throughput comparable to PBE-CC in walking cases. Learning-based approaches such as Aurora degrade by over 75% due to out-of-distribution dynamics. In Figure 9, we observe that PBE-CC achieves higher throughput than Aether but exhibits degraded latency performance. This validates that PBE-CC adjusts its cwnd strictly based on the physical-layer bandwidth. In fact, its feedback relies solely on the physical capacity without accounting for delay variation. Aether further introduces a penalty term for queue accumulation as shown in Equation 4, thereby strategically sacrificing a fraction of throughput to secure a significantly more stable latency performance.

High-bandwidth cellular networks. We further extract high-bandwidth traces from [19], which are available only in the specific scenarios with static and walking mobility patterns. For each trace, we conduct a 100-second emulation across all evaluated CCAs and record the corresponding network performance. Note that, due to the limited maximum uplink bandwidth in the available traces, the overall trace bandwidth currently does not exceed 200 Mbps. Figure 10 compares the evaluated CCAs under these high-bandwidth traces. The results show that Aether achieves excellent throughput and latency under high-bandwidth conditions, while other cellular-dedicated CCAs (e.g., PBE-CC, C2TCP, and Cesar) also exhibit comparable performance. Notably, due to the high uplink capacity and the trace-driven setup, minor network fluctuations are largely masked. Consequently, these cellular-specific CCAs cluster tightly on the graph, implying a marginal performance discrepancy. We intend to collect high-bandwidth uplink traces from real-world networks (including scenarios above 200 Mbps) in future work to conduct a more comprehensive study under realistic network dynamics.

Adaptation to network dynamics. Figure 11 illustrates the adaptation of CCAs under a highly dynamic from Belgium trace. The results show that Aether tightly tracks the rapidly fluctuating capacity envelope without significant overshoot or under-utilization. By leveraging direct driver-buffer feedback, Aether preemptively adjusts rates before packets accumulate in the bottleneck uplink. Delay-based methods, such as Copa, often under-utilize capacity due to jitter-induced false positives. Loss-based methods, such as CUBIC, react too slowly to bandwidth drops. More traces are provided in Appendix F.4.

6.4 Microbenchmarking

Fairness. We follow the fairness experimental setup of [54] by selecting three devices within the coverage of the same BS. These devices are connected to the same server during periods of low network load at midnight to eliminate background traffic interference, and data streams are initiated sequentially at 10-second intervals.

We report the fairness convergence process of each CCA in Figure 12. The fairness of CUBIC and Aurora has been sufficiently demonstrated in multiple studies [17, 22, 25, 35], and our test results can be referred to Appendix E.2. Each curve in the figure corresponds to one flow. Cesar's throughput shows significant fluctuations. C2TCP achieves high bandwidth utilization, but performance degrades in three-flow competition. Although BBRv3 achieves faster convergence, its bandwidth fluctuation is relatively severe. Copa+, which employs parameter adaptation, exhibits instability in cellular network scenarios and converges slowly (more than 20s) when three flows compete. In learning-based solutions, Vivace demonstrates better bandwidth utilization, yet its flow fluctuates wildly in competition. Astraea exhibits the worst bandwidth utilization performance, and its fairness is unsatisfactory. Aether quickly converges upon the arrival of new flows while maintaining efficient and stable bandwidth variations, demonstrating the best overall performance.

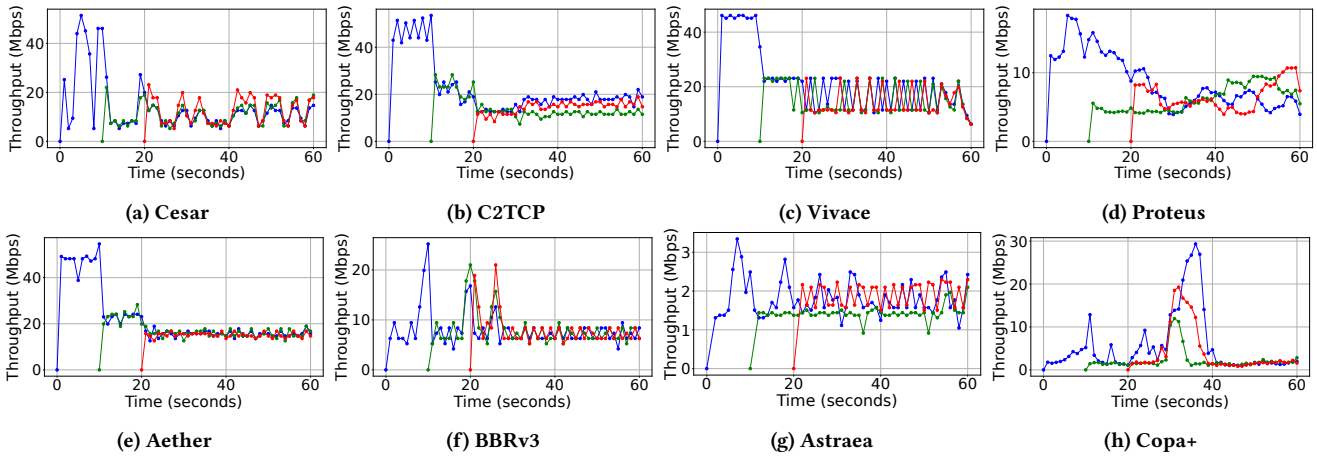


Figure 12: Throughput dynamics of different flows competing over the cellular uplink for various CCAs.

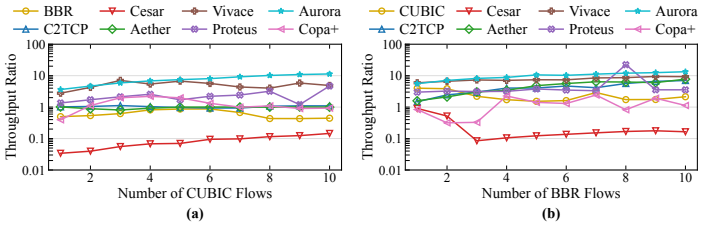


Figure 13: Throughput ratio to CUBIC and BBR of various CC schemes.

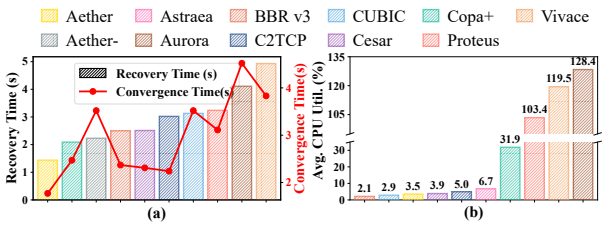


Figure 14: (a) Recovery and convergence time. (b) CPU utilization overhead.

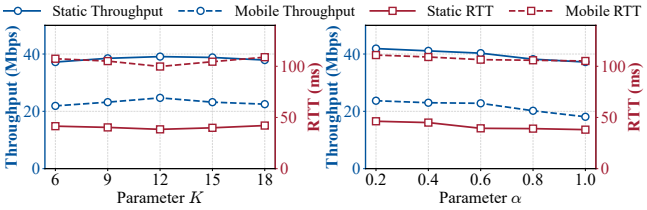


Figure 15: Parameter Analysis.

TCP friendliness. Figure 13 illustrates the average throughput ratios (evaluated CCA vs. legacy CCA) across varying numbers of competing flows in real-world cellular networks. As shown in Figure 13a, Aether demonstrates near-ideal strict fairness with CUBIC, consistently maintaining a throughput ratio around 1 regardless of the flow count, avoiding the starvation or severe degradation issues prevalent in other baselines. Conversely, when competing with BBR (Figure 13b), Aether achieves a throughput ratio strictly greater than 1. Importantly, this higher ratio does not imply that Aether is unfairly aggressive. As discussed previously, BBR fundamentally struggles with accurate bandwidth estimation in highly variable cellular environments, leaving the wireless capacity underutilized. Rather than starving competing BBR flows, Aether leverages its uplink capacity probing to efficiently reclaim and utilize this idle headroom, maximizing its own performance while ensuring benign coexistence.

Robustness to sudden network changes. We design an experiment based on moving across different conditions. We first operate UEs for 60 seconds in a good-signal location (RSRP of -90 dBm), then quickly move to a poor-signal location (RSRP of -110 dBm) with constant speed, and operate for 60 seconds. Then UEs travel back along the same trajectory and at the same speed to the good-signal location. We measure two key metrics over 100 repeated runs: ❶ **Convergence time**, time to converge to a stable rate (within 10% variance) after entering poor signal conditions; ❷ **Recovery time**, time to converge to a stable rate with 90% of its original steady-state level when back to good-signal location. Aether- indicates the Aether without the bottleneck detector and always controls the rate by Aether rate controller.

As shown in Figure 14a, Aether responds rapidly to sudden network changes, achieving a convergence time of just 1.44s and a recovery time of only 1.77s. By disabling the bottleneck detector in Aether-, the convergence and recovery times slow significantly to 3.52s and 2.23s, respectively. Notably, even without the Bottleneck Detector, Aether remains competitive with conventional algorithms.

Overhead. We also validate the computational overhead to UEs caused by CCAs. Each CCA runs continuously for 300 seconds under identical conditions, with detailed measurements of average CPU utilization recorded and presented in Figure 14b. CCAs are implemented in the Linux kernel space, such as CUBIC, BBR, C2TCP, Cesar, Aether, and demonstrates significantly lower CPU utilization. These algorithms benefit from their ability to achieve rapid responses by precisely processing, thereby avoiding unnecessary computational overhead. In contrast, Vivace, which also operates in kernel space, incurs substantially higher computational costs

due to its online learning-based control strategy that involves complex parameter adjustments and consequently prolongs the control loop cycle. Despite also being learning-based, Astraea demonstrates lower computational overhead, which is likely owing to its optimized C++ implementation.

Impact of parameters in our algorithm. Aether relies on K to send probing packets in the empty queue case, and α, γ for tuning the sending rate optimization. We evaluate the impact of selecting different values of these parameters. As shown in Figure 15, $K = 12$ and $\alpha = 0.6$ provide the best balance between throughput and latency. Aether exhibits low parameter sensitivity: performance varies by only 5.5%–12.6% across different settings. Even with suboptimal configurations, Aether still outperforms baselines by at least 18% in throughput and reduces latency by over 20%. More analysis and the impact of γ is in Appendix E.3.

Additional results to bottleneck detector. We further evaluate the bottleneck detector with additional tests, and provide the results in Appendix F.3. We also present the performance tests when cellular uplink is not the bottleneck in Appendix F.5 and showcase competitive performance of Aether.

7 Related Work

CCAs driven by end-to-end signals. CCA designs are constantly evolving to adapt to increasingly complex and diverse network conditions. Many of the CCAs [10, 12, 13, 22, 25, 26, 41, 53] use end-to-end signals such as RTT, packet loss to control the data sending rate, attempting to achieve optimal transmission performance across a wide variety of network conditions. Recently, an increasing number of learning-based CCAs [17, 24, 35, 40, 42, 51, 56, 61] have emerged. By using sophisticated models, these approaches leverage various learning methods, such as reinforcement learning (Astraea [35], Aurora [24]) and online learning (PCC Vivace [17], PCC Proteus [42]), aiming to generate control policies that are more suitable for the target network environment.

CCAs for wireless networks. In wireless networks, high-precision and fast-reacting transmission is often achieved by understanding radio-side operations. In relatively open and programmable WiFi environments, CCA solutions [32, 43] read the queue occupancy from the NIC/AP to optimize the sending rate. In contrast, cellular infrastructure is inherently a blackbox to CCAs. Studies [8, 31, 50, 53, 58] use transport-layer feedback to infer how UEs and BS operate. Representative works, such as C2TCP [8], mitigate wireless dynamics by estimating the minimum packet delay. Cesar [50] infers BS resource scheduling policy from TCP ACK inter-arrival times. The rapid fluctuations and complex low-layer operations in cellular networks make it difficult to accurately infer network conditions based purely on ACK arriving patterns, since the causes of latency are multifaceted. PEPesc [33] proposes an approach based on TCP performance enhancing proxies (PEPs) for cellular backhaul, which uses streaming coding to mask random packet losses and estimates available bandwidth to control the data rate.

CCAs driven by radio-side control signaling. CCAs such as ABC [20], PBE-CC [54], M2HO [38] access the cellular control signaling generated by the cellular subsystem. However, these works primarily focus on downlink scenarios, necessitating the use of ACKs to convey radio-side information back to the sender, which risks information obsolescence caused by network latency. Furthermore, the requirement for dual-sided modifications to convey ACK feedback, combined with the hardware dependencies for accessing control signaling, raises deployment concerns. Their research complements ours, as we pivot to the cellular uplink, a scenario that has become increasingly important today. Our insight reveals that in uplink transmission, CCAs should exploit the proximity advantage to capture cellular changes locally and instantaneously. This eliminates the dependency on ACK feedback and facilitates a design without the need for intrusive hardware modifications.

8 Conclusion

Unlike transmission over cellular downlink, the inefficiency of existing CCAs in the uplink stems not from a lack of visibility into cellular operations, but rather from the high latency and overhead involved in feeding these dynamics back to the CCAs. Simple but useful, Aether is proposed as an uplink-tailored CCA that promptly captures uplink dynamics via modem driver-buffer status, motivated by the insight that driver-level dwell time serves as an indicator of uplink congestion, and uses the rapid convergence to the optimal rate while sustaining consistently low latency, within highly dynamic network conditions. The primary goal of Aether is to achieve responsive, lightweight congestion control in resource-constrained uplink environments, without requiring any modifications beyond the sender's OS kernel. Extensive evaluations using both real-world experiments and trace-driven simulations confirm Aether's effectiveness in this regard. Aether provides an advantageous option that addresses the growing demand for high-performance transmission over cellular uplinks.

Our future work branches into several directions. First, we plan to conduct more comprehensive evaluations under extreme boundary conditions, such as high-frequency bottleneck transitions, alongside validating parameter robustness across diverse scenarios. Second, efforts will be directed toward extending Aether's deployment to a broader range of network drivers. Third, we aim to refine the design of cellular-dedicated trace-driven emulators, striving to align their replay fidelity closer with real-world network counterparts. Finally, we will explore fusing driver-buffer queue dynamics with upper-layer cellular telemetry to achieve a multi-dimensional awareness of cellular network status. Due to space constraints, a more comprehensive discussion and detailed elaboration on these future directions are provided in Appendix G.

Ethics: *This work does not raise any ethical issues.*

9 Acknowledgments

We sincerely thank our anonymous shepherd and all the SIGCOMM reviewers for their valuable feedback. This work is supported in part by the National Natural Science Foundation of China (No. 62471066, No. 62394323, No. 62572473, No. 62441230, No.62472240, No.562394322), the funding from the State Key Laboratory of Internet Architecture (No.HLW2025ZD17, No.HLW2025MS08), the Scientific Research Innovation Capability Support Project for Young Faculty (No. SRICSPYF-ZY2025001), Beijing Natural Science Foundation (L2603007), and China Postdoctoral Science Foundation under Grant No.2025M781467. Mu Wang and Yiying Lin have contributed equally to this work. Su Yao and Enhuan Dong are the corresponding authors.

References

[1] 2023. Android source code for MediaTek SoC cellular driver. https://github.com/MiCode/Xiaomi_Kernel_OpenSource/tree/ruby-s-oss/drivers/misc/mediatek/ccmni.

[2] 2024. Android source code for Qualcomm PCIe cellular driver. https://android.googlesource.com/kernel/msm/+/refs/tags/android-14.0.0_r0.140/drivers/bus/mhi.

[3] 2024. Android source code for Qualcomm SoC cellular driver. https://github.com/OnePlusOSS/android_kernel_modules_and_devicetree_oneplus_sm8750/blob/oneplus/sm8750_v_15.0.0_oneplus_13/vendor/qcom/opensource/dataipa/drivers/platform/msm/ipa/ipa_v3.

[4] 2025. *Android common kernels.*

[5] 2025. Android source code for MediaTek PCIe cellular driver. https://android.googlesource.com/kernel/common/+/refs/tags/android14-6.1-2025-02_r18/drivers/net/wwan/t7xx/.

[6] 2026. Aether: Code, Simulation Tools, and Trace Dataset. https://github.com/NextNetLab/Aether.

[7] 2026. Kernel overview. https://source.android.com/docs/core/architecture/kernel.

[8] Soheil Abbasloo, Yang Xu, and H Jonathan Chao. 2019. C2TCP: A flexible cellular TCP to meet stringent delay requirements. *IEEE Journal on Selected Areas in Communications* 37, 4 (2019), 918–932.

[9] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 632–647.

[10] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 329–342.

[11] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. 2019. Detecting if LTE is the Bottleneck with BurstTracker. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 1, 15 pages. doi:10.1145/3300061.3300140

[12] Lawrence S Brakmo, Sean W O'malley, and Larry L Peterson. 1994. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*. 24–35.

[13] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* 14, 5 (2016), 20–53.

[14] Chia-Yu Chang, Navid Nikaein, and Thrasyvoulos Spyropoulos. 2018. Radio access network resource slicing for flexible service execution. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 668–673.

[15] Hao Chen, Xu Zhang, Yiling Xu, Ju Ren, Jingtao Fan, Zhan Ma, and Wenjun Zhang. 2019. T-gaming: A cost-efficient cloud gaming system at scale. *IEEE Transactions on Parallel and Distributed Systems* 30, 12 (2019), 2849–2865.

[16] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath quic: Design and evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*. 160–166.

[17] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX symposium on networked systems design and implementation (NSDI 18)*. 343–356.

[18] Moinak Ghoshal, Imran Khan, Z Jonny Kong, Phuc Dinh, Jiayi Meng, Y Charlie Hu, and Dimitrios Koutsonikolas. 2023. Performance of cellular networks on the wheels. In *Proceedings of the 2023 ACM on Internet Measurement Conference*. 678–695.

[19] Moinak Ghoshal, Z. Jonny Kong, Qiang Xu, Zixiao Lu, Shivang Aggarwal, Imran Khan, Yuanjie Li, Y. Charlie Hu, and Dimitrios Koutsonikolas. 2022. An In-Depth Study of Uplink Performance of 5G mmWave Networks. In *Proceedings of 2nd ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases (5G-MeMU)*.

[20] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. 2020. {ABC}: A simple explicit congestion controller for wireless networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 353–372.

[21] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[22] Feixue Han, Qing Li, Peng Zhang, Gareth Tyson, Yong Jiang, Mingwei Xu, Yulong Lan, and ZhiCheng Li. 2024. {ETC}: An Elastic Transmission Control Using {End-to-End} Available Bandwidth Perception. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 265–284.

[23] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 1–10.

[24] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*. PMLR, 3050–3059.

[25] Xiang Ji, Changqiao Xu, Zekun Zhang, Lujie Zhong, Kai Gao, Han Xiao, Shujie Yang, and Gabriel-Miro Muntean. 2025. HydraCC: Finding the Pareto Frontiers of Congestion Control via Multi-objective Evolutionary Exploration. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*. IEEE, 1–10.

[26] Wanchun Jiang, Haoyang Li, Zheyuan Liu, Jia Wu, Jiawei Huang, Danfeng Shan, and Jianxin Wang. 2022. Copa+: Analysis and improvement of the delay-based congestion control algorithm copa. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 920–929.

[27] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. 2013. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 163–174.

[28] EMMETT Keeler and A Meir. 1969. A theorem on contraction mappings. *J. Math. Anal. Appl* 28, 1 (1969), 326–329.

[29] Imran Khan, Moinak Ghoshal, Joana Angjo, Sigrid Dimce, Mushahid Hussain, Paniz Parastar, Yenchia Yu, Xueting Deng, Sumit Hawal, Shirui Huang, et al. 2025. How mature is 5G deployment? A cross-sectional, year-long study of 5G uplink performance. *Computer Communications* 237 (2025), 108153.

[30] Jinsung Lee, Sungyong Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Hyoyoung Lim, Jihoon Lee, Xiaoqing Zhu, Sangeeta Ramakrishnan, Dirk Grunwald, Kyunghan Lee, et al. 2020. PERCEIVE: Deep learning-based cellular uplink prediction using real-time scheduling patterns. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 377–390.

[31] Wai Kay Leong, Zixiao Wang, and Ben Leong. 2017. TCP congestion control beyond bandwidth-delay product for mobile cellular networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 167–179.

[32] Lingang Li, Yongrui Chen, and Zhijun Li. 2024. QCC: Driver-Queue Based Congestion Control for Data Uploading in Wireless Networks. *IEEE Transactions on Mobile Computing* (2024).

[33] Ye Li, Liang Chen, Li Su, Kanglian Zhao, Jue Wang, Yongjie Yang, and Ning Ge. 2024. PEPesc: A TCP Performance Enhancing Proxy for Non-Terrestrial Networks. *IEEE Transactions on Mobile Computing* 23, 4 (2024), 3060–3076. doi:10.1109/TMC.2023.3269436

[34] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. 202–215.

[35] Xudong Liao, Han Tian, Chaoliang Zeng, Xinchen Wan, and Kai Chen. 2024. Astraea: Towards fair and efficient learning-based congestion control. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 99–114.

[36] Mufan Liu, Jie Chen, Gang Wu, Lei Ji, and Hao Wang. 2023. Soft-ack based outer loop link adaptation for latency-constrained 5g video conferencing. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 388–393.

[37] Yuxin Liu, Tianyang Zhang, Kyle Jamieson, and Yaxiong Xie. 2025. Seeing Through the Fog: Empowering Mobile Devices to Expose and Mitigate RAN Buffer Effects on Delay-Sensitive Protocols. *arXiv preprint arXiv:2507.00337* (2025).

[38] Zhutian Liu, Qing Deng, Zhaowei Tan, Zhiyun Qian, Xinyu Zhang, Ananthram Swami, and Srikanth V Krishnamurthy. 2024. M2HO: Mitigating the Adverse Effects of 5G Handovers on TCP. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1089–1103.

[39] Feng Lu, Hao Du, Ankur Jain, Geoffrey M Voelker, Alex C Snoeren, and Andreas Terzis. 2015. CQIC: Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. 45–50.

[40] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. 2022. Multi-objective congestion control. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 218–235.

[41] Stephen McQuistin, Colin Perkins, and Marwan Fayed. 2016. TCP Hollywood: An unordered, time-lined, TCP for networked multimedia applications. In *2016 IFIP networking conference (IFIP networking) and workshops*. IEEE, 422–430.

[42] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC proteus: Scavenger transport and beyond. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 615–631.

[43] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. 2022. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 193–206.

[44] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, et al. 2021. A variegated look at 5G in the wild: performance, power, and QoE implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*.

610–625.

[45] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 417–429.

[46] Ashkan Nikravesh, Yihua Guo, Feng Qian, Z Morley Mao, and Subhabrata Sen. 2016. An in-depth understanding of multipath TCP on mobile devices: Measurement and system design. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. 189–201.

[47] Lianyong Qi, Xiaolong Xu, Xiaotong Wu, Qiang Ni, Yuan Yuan, and Xuyun Zhang. 2023. Digital-twin-enabled 6g mobile network video streaming using mobile crowdsourcing. *IEEE Journal on Selected Areas in Communications* 41, 10 (2023), 3161–3174.

[48] I Rhee, L Xu, S Ha, A Zimmermann, L Eggert, and R Scheffenegger. 2018. Rfc 8312: Cubic for fast long-distance networks.

[49] William Sentosa, Balakrishnan Chandrasekaran, P Brighten Godfrey, and Haitham Hassanieh. 2025. {CellReplay}: Towards accurate record-and-replay for cellular networks. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 1169–1186.

[50] Juhun Shin, Goodsol Lee, Jeongyeup Paek, and Saewoong Bahk. 2025. César: Cellular Resource Scheduling-Aware Congestion Control. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*. IEEE, 1–10.

[51] Han Tian, Xudong Liao, Decang Sun, Chaoliang Zeng, Yilun Jin, Junxue Zhang, Xinchen Wan, Zilong Wang, Yong Wang, and Kai Chen. 2025. Achieving Fairness Generalizability for Learning-based Congestion Control with Jury. In *Proceedings of the Twentieth European Conference on Computer Systems*. 413–427.

[52] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180.

[53] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 459–471.

[54] Yaxiong Xie, Fan Yi, and Kyle Jamieson. 2020. PBE-CC: Congestion control via endpoint-centric, physical-layer bandwidth measurements. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 451–464.

[55] Yin Xu, Zixiao Wang, Wai Kay Leong, and Ben Leong. 2014. An end-to-end measurement study of modern cellular data networks. In *International Conference on Passive and Active Network Measurement*. Springer, 34–45.

[56] Chen-Yu Yen, Soheil Abbasloo, and H Jonathan Chao. 2023. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 255–274.

[57] Fan Yi, Oliver Michel, Haoran Wan, and Kyle Jamieson. 2024. Understanding the Impact of Cellular RAN-induced Delay on Video Conferencing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1641–1643.

[58] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 509–522.

[59] Pengfei Zhang, Dingzhu Wen, Guangxu Zhu, Qimei Chen, Kaifeng Han, and Yuanming Shi. 2024. Collaborative edge AI inference over cloud-RAN. *IEEE Transactions on Communications* 72, 9 (2024), 5641–5656.

[60] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. 2021. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 418–432.

[61] Zhiren Zhong, Wei Wang, Yiyang Shao, Zhenyu Li, Heng Pan, Hongtao Guan, Gareth Tyson, Gaogang Xie, and Kai Zheng. 2022. Muses: Enabling lightweight learning-based congestion control for mobile devices. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2208–2217.