

Characterizing the Impact of Active Queue Management on Speed Test Measurements

Siddhant Ray
University of Chicago

Taveesh Sharma
University of Chicago

Jonatas Marques
University of Chicago

Paul Schmitt
Cal Poly

Francesco Bronzino
ENS Lyon

Nick Feamster
University of Chicago

Abstract

Present day speed test tools measure peak throughput, but often fail to capture the user-perceived responsiveness of a network connection under load. Recently, platforms such as NDT, Ookla Speedtest and Cloudflare Speed Test have introduced metrics such as “latency under load” or “working latency” to fill this gap. Yet, the sensitivity of these metrics to basic network configurations such as Active Queue Management (AQM) remains poorly understood. In this work, we conduct an empirical study of the impact of AQM on speed test measurements in a laboratory setting. Using controlled experiments, we compare the distribution of throughput and latency under different load measurements across different AQM schemes, including CoDel, FQ-CoDel and Stochastic Fair Queuing (SFQ). On comparing with a standard drop-tail baseline, we find that measurements have high variance across AQM schemes and load conditions. These results highlight the critical role of AQM in shaping how emerging latency metrics should be interpreted, and underscore the need for careful calibration of speed test platforms before their results are used to guide policy or regulatory outcomes.

1 Introduction

Internet performance has historically been summarized using a single number: “speed” [5, 12, 27]. Despite their widespread utility, traditional measurements that focus on peak bandwidth (*i.e.*, link capacity) fail to capture user-perceived Quality of Experience (QoE). For many interactive applications (e.g., remote collaboration, gaming, video conferencing), QoE is determined not by the maximum attainable rate but by the available bandwidth and latency under load, which conventional full-capacity speed tests do not accurately reflect. However, the interpretation and use of these metrics has not been standardized. For instance, NDT [26], reports average throughput and some aggregated measurements of total bytes transferred during the test. Ookla defines “working latency” as the increase in round-trip time (RTT) under load compared to the unloaded RTT, measured during a speed test [11]. Apple uses a different metric, called round trips per minute (RPM) under load, which counts the number of round trips completed during a fixed time interval

while the connection is saturated [32]. Further, these tests have been known to discard outliers that often correspond to glitches that users typically notice during real-time applications such as video conferencing and streaming [11]. As a result, users and regulators are left with incomplete pictures of what causes an Internet connection to be unresponsive, and how it can be mitigated. A central, unanswered question is how traditional metrics such as throughput and latency, and new metrics such as LUL, behave in the presence or absence of active queue management (AQM) algorithms such as FQ-CoDel, which were explicitly designed to maintain low latency under load [16].

While AQM has been widely studied in the context of TCP performance, its impact on speed test measurements remains underexplored. Given that AQM deployment is steadily increasing¹, understanding this relationship is crucial for both network operators and end-users to accurately assess and improve their internet experience. In this paper, we investigate how the empirical distribution of modern speed test measurement results shifts when an AQM is deployed. Rather than reporting only typical throughput (e.g., mean) and latency values (e.g., 90th percentile, median), we analyze full distributions: the tails, the spikes, and metrics similar to “glitches per minute” [11] that are most relevant to real-time applications. Our goal is to empirically characterize the difference between unmanaged queues and AQM-enabled network paths, and to highlight how this difference is (or is not) reflected in widely deployed measurement platforms. By doing so, we aim to inform both test designers and network operators of the gaps between the status quo of Internet measurement and the actual experience of end-users.

In summary, our contributions are as follows:

- We conduct a measurement study with a popular speed test tool (NDT) [26] across varying AQM algorithms and network conditions to analyze impact of such policies and link load on speed test measurements.
- We show that traditional speed test metrics such as averaged throughput and latency can mask significant variations in user-perceived speed and responsiveness and

¹Based on conversations with leading ISP vendors in the United States.

believe that our findings should help inform the design of future speed test tools should consider network conditions and AQM policies to report more interpretable metrics.

2 Background

2.1 Active Queue Management (AQM)

AQM techniques have been an active area of research and deployment for the past few decades to reduce latency and bufferbloat [15] and to ensure fairness and coexistence among TCP flows on the shared network links. Traditional AQM algorithms have been built to run in conjunction with TCP congestion control algorithms, which rely only on packet loss as a signal for congestion.

To handle the bursty nature of TCP, these AQM techniques are equipped with large data buffers to prevent excessive packet drops due to these bursts. However, bufferbloat arises when queue length grows unbounded, specifically if the buffers are increasingly large, the packets end up being queued in much deeper queues, leading to excessive queuing latency. latency can often build up as a result of individual bufferbloats at multiple routers on the network path.

AQM techniques such as Random Early Detection [14], CoDel [30] and FQ-CoDel [18] have been designed to reduce latency and bufferbloat, by actively managing the queue lengths by dropping packets before the queue becomes full. Other techniques such as Stochastic Fair Queuing (SFQ) [24] and Deficit Round Robin (DRR) [43] aim to provide fairness among competing flows by ensuring equal bandwidth allocation. AQM techniques such as Proportional Integral controller Enhanced (PIE) [33], CAKE [19] and Low Latency, Low Loss, Scalable Throughput (L4S) [41] have been proposed to provide low latency and high throughput for modern applications such as video streaming and online gaming.

These recent advances in AQM suggest that integrating design principles bufferbloat and fairness (e.g., CAKE and L4S), which can yield significant performance improvements over other traditional AQM techniques. CAKE emphasizes deployability in heterogeneous consumer environments by incorporating bandwidth shaping, per-flow and per-host fairness, DiffServ awareness, and optional TCP ACK filtering to mitigate asymmetry and NAT-induced complexity. L4S introduces a congestion control framework based on Explicit Congestion Notification (ECN) signaling and shallow queuing, enabling sub-millisecond latency and high throughput when paired with responsive transport protocols.

These AQM techniques are being widely adopted in various network devices [39, 47], including home routers, enterprise networks, and data centers, to improve network performance and user experience. However, measuring the impact of AQM techniques by speed test measurement tools remains

underexplored due to the lack of standardized methodologies and metrics to evaluate their overall performance.

2.2 Speed Test Measurement Tools

A variety of speed test measurement tools are commonly employed to assess end-to-end network performance. The Measurement Lab’s Network Diagnostic Tool (NDT) [25, 26] is a widely deployed open platform that enables broadband quality evaluation through standardized TCP-based tests. NDT establishes a controlled client-server connection to measure achievable throughput, round-trip latency, and packet loss, and in its most recent version (NDT7) uses WebSocket transport for compatibility with modern browsers. In addition to providing a user-facing interface, NDT publishes all test results as open data, thereby supporting reproducible, large-scale research on global internet performance.

Ookla’s speed test [1, 31] is a commercial measurement service that conducts real-time assessments of download and upload speeds, latency, and jitter. It typically operates by initiating multiple parallel TCP flows between the client and one of its geographically distributed test servers, thereby approximating the aggregate throughput available to a user under normal usage conditions. The service automatically selects the closest or least-loaded server to minimize measurement bias, and results are recorded through an interactive interface widely deployed across web and mobile platforms.

Apple’s speed test [3] application is designed for iOS devices and provides a lightweight interface for evaluating network performance directly from mobile hardware. The tool performs measurements of download and upload rates and latency using a set of Apple-operated servers integrated into the broader iOS ecosystem. By leveraging native APIs and mobile-specific optimizations, the application captures performance metrics in a context that reflects real-world mobile usage scenarios.

Fast.com [29], developed by Netflix, is a web-based measurement service that focuses primarily on download speed evaluation, reflecting the bandwidth available for streaming media. The test runs automatically when a user visits the site, initiating multiple parallel HTTP-based flows to Netflix-operated content servers. The service also provides optional measurements of upload speed and latency under load, offering a lightweight, consumer-oriented approach to performance testing of speed test measurement.

These tools collectively provide standardized yet distinct methodological approaches for assessing network quality, supporting both individual diagnostics and broader internet performance research. However, they share a common limitation in that they do not measure latency under sustained load conditions. Latency under load is crucial for characterizing user-visible network performance as it reveals queueing

delays and congestion behavior. Existing speed test tools primarily report peak throughput and baseline RTT, which underrepresents performance during sustained load, and provide an incomplete view of network quality and systematically misestimate real-world experience.

2.3 Related Work

speed test measurement research: Several studies have explored methodologies and challenges in measuring broadband performance using diverse tools and datasets [4, 6, 7, 36, 37, 44, 45]. The authors behind the M-Lab platform have provided an open, large-scale testbed for broadband diagnostics, enabling researchers to analyze throughput, latency, and access network bottlenecks across user populations [7, 45]. Follow-up studies have leveraged M-Lab data to understand end-to-end performance and characterize latency dynamics in last-mile access networks [6, 36]. In parallel, commercial systems such as Ookla’s speed test have been examined for large-scale performance estimation and for highlighting biases in crowdsourced measurement datasets [4]. More recent work has analyzed web-based speed test platforms such as Netflix’s Fast.com, uncovering methodological differences and identifying how test design influences user-visible metrics [44]. Complementary analyses of measurement bias and representativeness in crowdsourced platforms emphasize the need for careful calibration when interpreting speed test-derived metrics [37].

3 Methodology

In order to study the impact of active queue management (AQM) on speed test results we set up an isolated testbed. Our testbed is composed of two System76 Meerkat 6 mini computer hosts connected together with a Turris Omnia open-source router. Both Meerkat mini computers are configured with an Intel Core i5-1135G7 (2.4GHz) processor, 16GB of DDR4 RAM (3200 MHz), a 2.5GbE-capable Intel Ethernet Controller I225-LM, and Ubuntu 20.04.5 LTS (GNU/Linux 5.18.10-76051810-generic x86_64). The Omnia router is configured with Marvell Armada 385 (1.6GHz), 2GB of DDR3 RAM, five 1GbE LAN ports, one 1GbE WAN port, and TurrisOS 7.2.3 (GNU/Linux 5.15.148).

With this 1GbE-capable testbed, we perform a series of speed tests to collect results in many specific scenarios varying a number of options and conditions. Namely, we test different (a) speed test tools, (b) AQM algorithms, (c) throughput limits, (d) base latency values, (e) usage of burst shaping for maximum link utilization, and (f) presence or absence of competing traffic.

Regarding speed testing, we focus on two tools: Measurement Lab Network Diagnostic Tool (NDT) and iperf3 based tests for TCP. We have chosen these two tools due to their

scale and popularity, in terms of number of vantage points, size of datasets, and their usage in research and policy communities [2, 8–10, 17, 20, 21, 23, 28, 34, 35, 38, 40, 42, 46]. For both speed test tools, we designate one of our hosts as the server, installing the necessary server-side application, and run tests pointing to the server from the other host using command-line client applications. We configure the speed test clients to output verbose results and save all data for post-hoc analysis. Finally as NDT does not report instantaneous throughput measurements, we also collect pcaps to measure instantaneous throughput values.

We evaluate three individual AQM algorithms: CoDel (Controlled Delay), FQ-CoDel (Fair/Flow Queue CoDel), and SFQ (Stochastic Fair Queuing). We also run tests with the default queuing discipline, i.e., drop-tail, which we refer to as No AQM. These algorithms were chosen for their direct availability in most Linux installations and are configured in the testbed router with their respective default parameters on TurrisOS Linux using the `tc qdisc` command.

For throughput limits, we consider the values between 100 Mbps and 1Gbps in steps of 100 Mbps, representing popular modern residential speeds. Similar to AQM algorithm, we configure rate limiting using `tc` and the Hierarchy Token Bucket algorithm in the router. We set the rate parameter to the respective throughput limit and we evaluate the setups with and without burst shaping enabled. When burst shaping is enabled, we set the `ceil` parameter to $1.6 \times$ the base rate, the `burst` parameter to 15KB, and the `cburst` parameter to 30KB. The respective values for `ceil` was chosen based on the common practice of setting `ceil` = $1.2 \times$ to $2 \times$ rate depending on expected link utilization and burstiness, making $1.6 \times$ a balanced heuristic between fairness and flexibility [13]. For the `burst` and `cburst` parameters, we follow the practices suggested in the `tc` manual [22], which recommend setting these values to accommodate for few hundred milliseconds of bursty traffic. This allows us enough burst sustain TCP data transmission and avoid jitter without getting excessively aggressive at the given shaping rate.

We also experiment with introducing artificial delay to the interface connecting hosts in order to emulate the typical communication latency on the Internet. Initially, we configure the router interface to introduce such delay using the `tc netem` command. However, in our testing we detect that the CPU-intensive nature of this algorithm saturates the router and makes performance highly unstable. To address this, we allow for optionally adding the delay in the server with a `netem` discipline acting over the outgoing traffic using Linux `tc` and IFB (Intermediate Functional Block) device. Since Linux `tc netem` can only shape outbound traffic, an IFB device is used to redirect incoming packets so they can be shaped as if they were outgoing. However, we observed

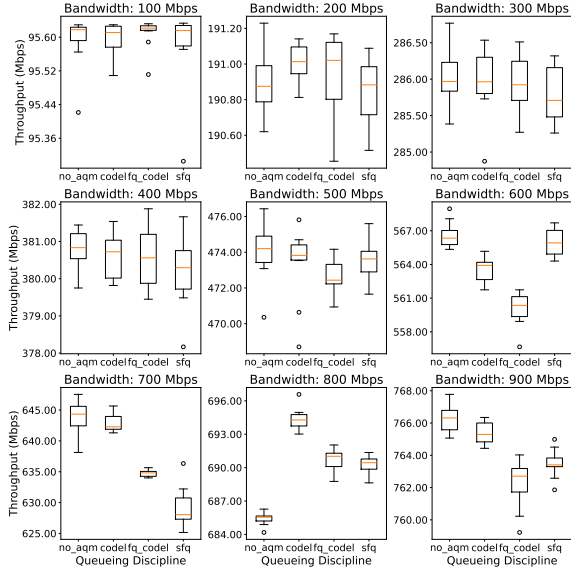


Figure 1: Throughput measurements without burst shaping and without cross traffic has lower variability across different AQM algorithms but often underutilizes the link capacity.

that adding such latency merely offsets the latency measurements by a constant factor, without affecting throughput or relative latency under load. Therefore, in this paper we only present results without added latency.

Finally, we evaluate the effect of competing traffic in the path between client and server hosts. We introduce competing traffic by running the `iperf3` tool at the same time as the speed tests. We use TCP Cubic to add cross traffic as it is the default congestion control algorithm in Linux. Our experimental setup consists of starting an (indefinitely-running) `iperf3` test 5 seconds before launching the speed test tool and once the speed test ends, we also terminate the `iperf3` test. We allow the cross traffic to be capacity-seeking (we do not set a target rate) to use all available bandwidth.

4 Speed Test Measurement Study Results

4.1 End-to-end measurement results reported with NDT

For all the results presented in this section, we run NDT speed tests on our lab setup as introduced in the Section 3. We measure speed tests across all AQM algorithms and every test is repeated 10 times. We refer to the AQM methods with the labels `no_aqm`, `code1`, `fq_code1`, and `sfq` for the No AQM, CoDel, FQ-CoDel, and SFQ AQM algorithms respectively. In our terminology, `no_aqm` refers to `pfifo` which is the default drop-tail queueing discipline in Linux.

NDT speed tests on the lab testbed without burst shaping or TCP Cubic cross traffic: We use our lab setup as

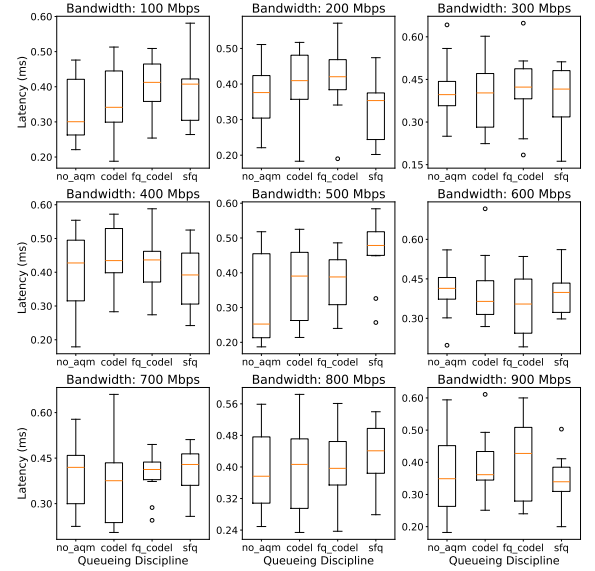


Figure 2: Latency measurements without burst shaping and without cross traffic does not introduce significant latency spikes due to low of queuing delays.

introduced in the Section 3 to run NDT speed tests without explicitly adding cross traffic. This represents a baseline scenario where the only traffic is from the speed test measurement tool and the measurements are not performed under load. We do not introduce any link latency using `netem` for this measurement. Figures 1 and 2 show the throughput and latency measurements respectively.

We observe the distribution of the measurements have measureable variations across the AQM algorithms used, even if the mean values don't report larger variations. Furthermore, if we do not use burst shaping, we observe that the bandwidth of the shaped link can be significantly underutilized, especially for higher rate limits (e.g., At bandwidth of 800 Mbps, the maximum link utilization is less than 700 Mbps) which affects the throughput measurements.

NDT speed tests on the lab testbed with burst shaping without TCP Cubic cross traffic: For these measurements we run NDT speed tests on our lab setup by introducing burst shaping on the link but without adding any cross traffic. The goal of this scenario is to evaluate the measurements by maximizing the link utilization for the speed test tool. This reflects measurements of these tools when the data transfer from the test itself is maximized, without additional competing traffic.

Figures 3 and 4 show the throughput and latency measurements respectively. We observe as opposed to the case without burst shaping, the throughput measurements are closer to the configured rate limits for the shaped link (e.g., At bandwidth of 800 Mbps, the maximum link utilization is now close to 800 Mbps). We conclude burst allowance is

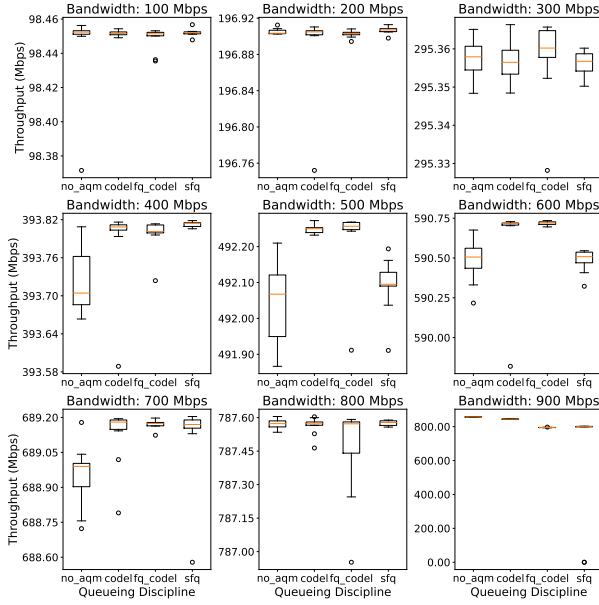


Figure 3: Throughput measurement with burst shaping without cross traffic achieve higher link utilization and better use of available bandwidth across all AQM algorithms.

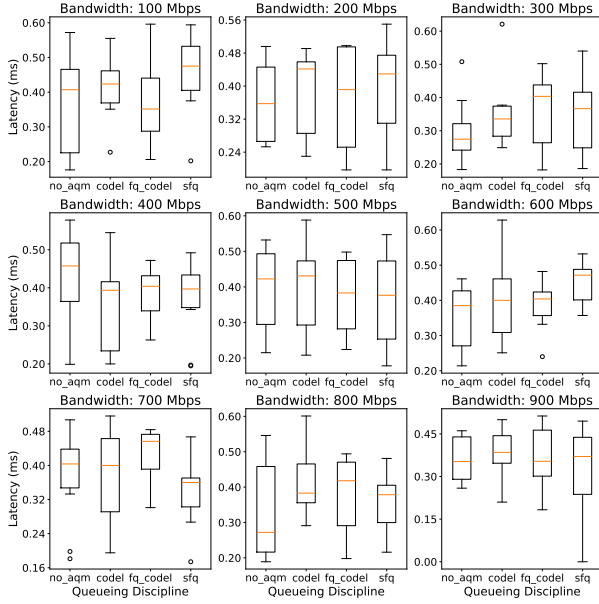


Figure 4: Latency measurement with burst shaping without cross traffic is still stable due to lower competing load on the link.

necessary to allow speed test tools to maximize link utilization and accurately measure throughput, especially at higher bandwidths.

NDT speed tests on the lab testbed with burst shaping and TCP Cubic cross traffic: For these measurements we

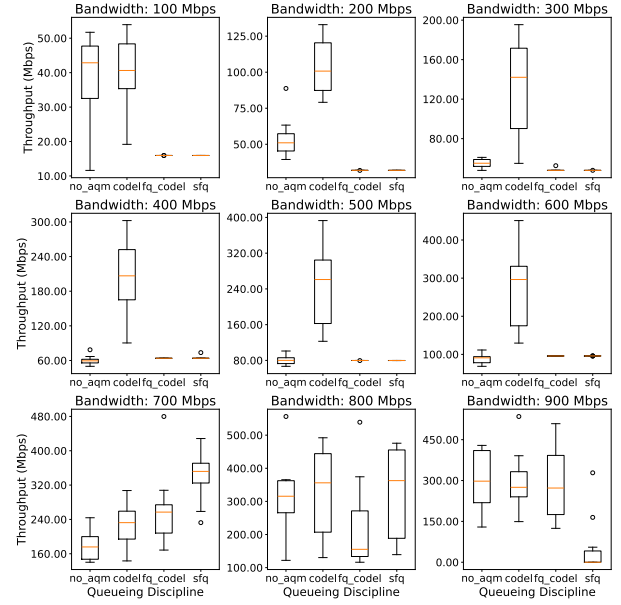


Figure 5: Throughput measurement with burst shaping and TCP cross traffic shows higher variability across different AQM algorithms as these become more prominent under load from competing traffic.

run NDT speed tests on our lab setup with TCP Cubic as cross traffic. This represents a scenario where the speed test measurements are performed under load from competing TCP traffic. TCP Cubic cross traffic is generated by running an iperf3 test in parallel with the NDT speed test as mentioned in the section 3. The TCP flows are set to be capacity seeking and try to use all available bandwidth. We do not introduce any link latency using netem on the server side. Finally, burst shaping is enabled on the link so that the speed test tool is able to maximize the link utilization.

We observe from Figure 5 that the throughput measurements are significantly lowered due to the external load from the TCP Cubic cross traffic. The throughput measurements are also significantly more variable across all AQM algorithms as compared to the previous two scenarios without cross traffic (e.g., for fq_code1 at bandwidth 900 Mbps, the standard deviation is 138 Mbps with cross traffic and burst shaping but 1.51 Mbps without both). We also observe in Figure 6 that the latency measurements are significantly higher due to the queuing delays introduced by the cross traffic. We conclude that interpretation of speed test measurements by operators should consider the effect of network conditions and AQM policies in order to accurately interpret the results beyond a time-averaged value over the duration of the test.

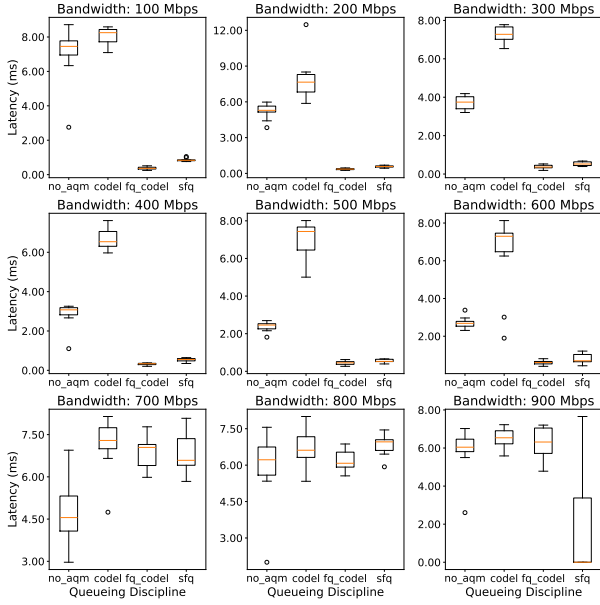


Figure 6: Latency measurement with burst shaping and TCP cross traffic shows significant increase in latency due to queuing delays introduced by competing traffic.

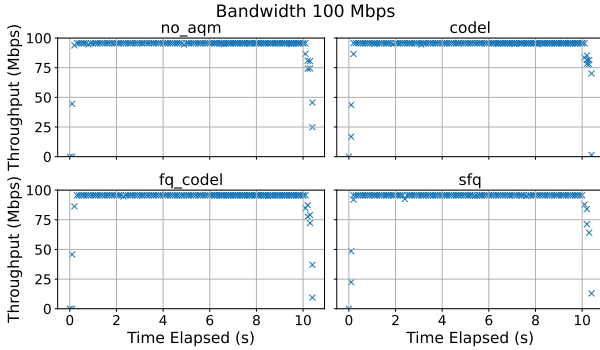


Figure 7: Instantaneous upload throughput measurement without burst shaping or cross traffic at low bandwidth (100 Mbps) does not harness the characteristics of different AQM policies.

4.2 Instantaneous measurement results reported with NDT

We further study the variability of instantaneous throughput during the NDT speed tests by analyzing the time series data of upload throughput. This allows us to observe how throughput changes over time, particularly in response to network conditions and competing traffic. We first consider a baseline scenario without burst shaping or cross traffic as shown in Figure 7. We see that the instantaneous throughput remains relatively stable across all of the AQM algorithms with minimal variability, as also observed in Figure 1. In

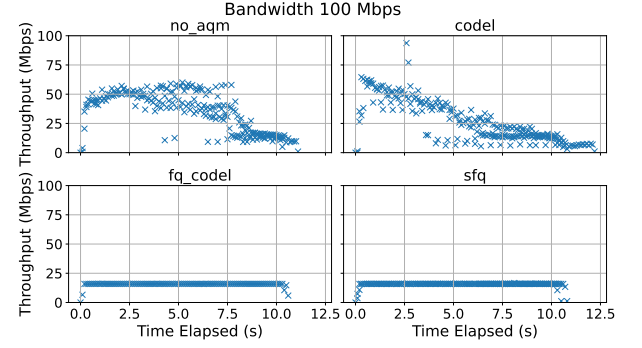


Figure 8: Instantaneous upload throughput measurement with burst shaping and cross traffic at low bandwidth (100 Mbps) shows some AQMs are better at maintaining stable throughput.

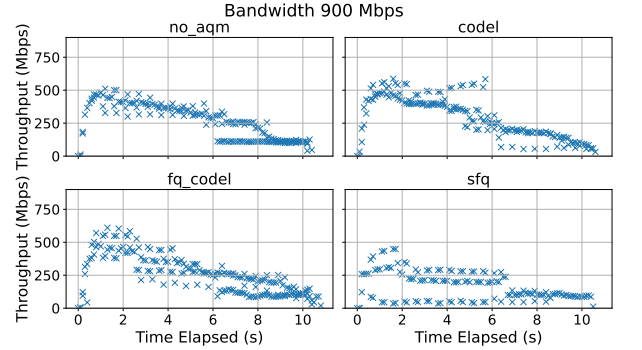


Figure 9: Instantaneous upload throughput measurement with burst shaping and cross traffic at high bandwidth (900 Mbps) shows significant variability across all AQM algorithms and affects the stability of throughput measurements.

measurements conducted under low load, the choice of AQM algorithm has minimal impact on throughput stability.

Next we analyze the instantaneous throughput with burst shaping and TCP Cubic cross traffic enabled. We compare two scenarios: one with low bandwidth limit (100 Mbps) and another with high bandwidth limit (900 Mbps). In both scenarios, we enable burst shaping and introduce TCP Cubic cross traffic. We see at lower bandwidth limits, the instantaneous throughput remains relatively stable across two of AQM algorithms (fq_code1 and sfq) as shown in Figure 8. However, at higher bandwidth limits, the instantaneous throughput exhibits significant variability across all AQMs as shown in Figure 9. Hence, measuring the impact of load and AQM policies on the stability and variability of speed test tools is crucial for accurate interpretation of the results.

5 Conclusion

We present a measurement study analyzing the impact of AQM algorithms on speed test measurement tools, focusing on variability across different AQM policies and network conditions. We show that speed test measurements can exhibit significant variability across test duration and reporting only aggregated values misses important dynamics of network performance. Hence, interpretation of speed test measurements should be aware of these underlying network factors while the tests themselves may be evolved to report more fine-grained metrics.

References

- [1] 2023. Understanding Speedtest Methodology. <https://www.ookla.com/articles/ookla-speedtest-methodology>. Whitepaper, Accessed: 2025-10-02.
- [2] Dale Alexander and Lika Döhl Diouf. 2022. Illustrating Internet Speed Divides in the Caribbean During COVID-19. *FOCUS Magazine of the Caribbean Development and Cooperation Committee (CDCC)* (12 2022). <https://hdl.handle.net/11362/48956>
- [3] Apple Inc. 2025. Apple Speedtest. <https://support.apple.com/HT212313>. Accessed: 2025-10-02.
- [4] Babak Arzani, Theophilus Benson, David Maltz, and Lucian Popa. 2016. Speedtest at Scale: Measurement and Biases in Crowdsourced Performance Data. In *Proceedings of the ACM SIGMETRICS Conference*. ACM, 307–318. doi:10.1145/2896377.2901480
- [5] Steven Bauer, David D Clark, and William Lehr. 2010. Understanding broadband speed measurements. TPRC.
- [6] Zachary S. Bischof, André Callado, Srikanth Sundaresan, and Nick Feamster. 2019. Characterizing latency in broadband access networks. In *Proceedings of the Passive and Active Measurement Conference (PAM)*. Springer, 65–77. doi:10.1007/978-3-030-15986-3_5
- [7] Zdravko Bozakov, Aaron Schulman, and Srikanth Sundaresan. 2018. M-Lab: An Open Platform for Large-Scale Network Measurement. In *Proceedings of the ACM SIGCOMM Conference (Demo Session)*. ACM, 80–81. doi:10.1145/3234200.3234239
- [8] Center for Rural Pennsylvania. 2018. Broadband Availability and Access. <https://www.rural.pa.gov/publications/broadband.cfm> Accessed November 13, 2024.
- [9] David D Clark and Sara Wedeman. 2021. Measurement, meaning and purpose: Exploring the M-Lab NDT dataset. In *TPRC49: The 49th Research Conference on Communication, Information and Internet Policy*. <https://dx.doi.org/10.2139/ssrn.3898339>
- [10] David D Clark and Sara Wedeman. 2024. Measurement of Internet access latency: A cross-dataset comparison. In *Proceedings of the TPRC2024 The Research Conference on Communications, Information and Internet Policy*. <http://dx.doi.org/10.2139/ssrn.4909679>
- [11] Dave Täht. [n. d.]. The new features and flaws of speedtest.net, ookla and cloudflare. <https://blog.cerowrt.org/post/speedtests/>. Accessed: 2025-10-01.
- [12] Nick Feamster and Jason Livingood. 2019. Internet Speed Measurement: Current Challenges and Future Recommendations. arXiv:1905.02334 [cs.NI] <https://arxiv.org/abs/1905.02334>
- [13] S. Floyd. 2001. A report on recent developments in TCP congestion control. *Comm. Mag.* 39, 4 (April 2001), 84–90. doi:10.1109/35.917508
- [14] S. Floyd and V. Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (1993), 397–413. doi:10.1109/90.251892
- [15] Jim Gettys and Kathleen Nichols. 2011. Bufferbloat: Dark Buffers in the Internet: Networks without effective AQM may again be vulnerable to congestion collapse. *Queue* 9, 11 (nov 2011), 40–54. doi:10.1145/2063166.2071893
- [16] Toke Høiland-Jørgensen. [n. d.]. Bufferbloat and Beyond. ([n. d.]).
- [17] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. 2016. Measuring Latency Variation in the Internet. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (Irvine, California, USA) (CoNEXT '16). Association for Computing Machinery, New York, NY, USA, 473–480. doi:10.1145/2999572.2999603
- [18] Toke Høiland-Jørgensen, Paul Mc Kenney, dave.taht@gmail.com, Jim Gettys, and Eric Dumazet. 2018. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290. doi:10.17487/RFC8290
- [19] Toke Høiland-Jørgensen, Dave Täht, and Jonathan Morton. 2018. Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways. arXiv:1804.07617 [cs.NI] <https://arxiv.org/abs/1804.07617>
- [20] Hanyang Jiang, Henry Shaowu Yuchi, Elizabeth Belding, Ellen Zegura, and Yao Xie. 2023. Mobile Internet Quality Estimation using Self-Tuning Kernel Regression. arXiv:2311.05641 [stat.AP] <https://arxiv.org/abs/2311.05641>
- [21] Hyeonseong Lee, Udit Paul, Arpit Gupta, Elizabeth Belding, and Mengyang Gu. 2023. Analyzing Disparity and Temporal Progression of Internet Quality through Crowdsourced Measurements with Bias-Correction. arXiv:2310.16136 [stat.AP] <https://arxiv.org/abs/2310.16136>
- [22] Linux man-pages project. 2025. tc - traffic control. <https://man7.org/linux/man-pages/man8/tc.8.html> Accessed through 'man tc' on a Linux system.
- [23] Kyle MacMillan, Tarun Mangla, James Saxon, Nicole P. Marwell, and Nick Feamster. 2023. A Comparative Analysis of Ookla Speedtest and Measurement Labs Network Diagnostic Test (NDT7). *Proc. ACM Meas. Anal. Comput. Syst.* 7, 1, Article 19 (March 2023), 26 pages. doi:10.1145/3579448
- [24] Paul E. Mc Kenney. 1990. Stochastic Fairness Queuing. In *IEEE INFOCOM'90 Proceedings*. The Institute of Electrical and Electronics Engineers, Inc., San Francisco, 733–740.
- [25] Measurement Lab. 2025. NDT7 Protocol Specification. <https://github.com/m-lab/ndt-server/blob/main/spec/ndt7-protocol.md>. Accessed: 2025-10-02.
- [26] Measurement Lab. 2025. Network Diagnostic Tool (NDT). <https://www.measurementlab.net/tests/ndt/>. Accessed: 2025-10-02.
- [27] Cise Midoglu, Leonhard Wimmer, Andra Lutu, Ozgu Alay, and Carsten Griwodz. 2018. MONROE-Nettest: A Configurable Tool for Dissecting Speed Measurements in Mobile Broadband Networks. arXiv:1710.07805 [cs.NI] <https://arxiv.org/abs/1710.07805>
- [28] Syed Tauhidun Nabi, Zhuowei Wen, Brooke Ritter, and Shaddi Hasan. 2024. Red is Sus: Automated Identification of Low-Quality Service Availability Claims in the US National Broadband Map. In *Proceedings of the 24th ACM on Internet Measurement Conference (IMC '24)*. Association for Computing Machinery, New York, NY, USA, 2–18. doi:10.1145/3646547.3688441
- [29] Netflix. 2025. Fast.com Internet Speed Test. <https://fast.com/>. Accessed: 2025-10-02.
- [30] Kathleen Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (jul 2012), 42–50. doi:10.1145/2209249.2209264
- [31] Ookla. 2025. Speedtest by Ookla. <https://www.speedtest.net/>. Accessed: 2025-10-02.
- [32] Christoph Paasch, Randall Meyer, Stuart Cheshire, and Will Hawkins. 2025. Responsiveness under Working Conditions. Internet-Draft draft-ietf-ippm-responsiveness-07. Internet Engineering Task Force. <https://www.ietf.org/archive/id/draft-ietf-ippm-responsiveness-07.html>

- //datatracker.ietf.org/doc/draft-ietf-ippm-responsiveness/07/ Work in Progress.
- [33] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*. 148–155. doi:10.1109/HPSR.2013.6602305
- [34] Udit Paul, Jiamo Liu, David Farias-Ilerenas, Vivek Adarsh, Arpit Gupta, and Elizabeth Belding. 2022. Characterizing Internet Access and Quality Inequities in California M-Lab Measurements. In *Proceedings of the 5th ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies* (Seattle, WA, USA) (COMPASS '22). Association for Computing Machinery, New York, NY, USA, 257–265. doi:10.1145/3530190.3534813
- [35] Udit Paul, Jiamo Liu, Mengyang Gu, Arpit Gupta, and Elizabeth Belding. 2022. The importance of contextualization of crowdsourced active speed test measurements. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) (IMC '22). Association for Computing Machinery, New York, NY, USA, 274–289. doi:10.1145/3517745.3561441
- [36] Aditya Rao, Arpit Gupta, Nick Feamster, and Balachander Krishnamurthy. 2020. Understanding Bottlenecks in the Internet Last Mile. In *Proceedings of the ACM Internet Measurement Conference (IMC)*. ACM, 321–335. doi:10.1145/3419394.3423642
- [37] Juan Restrepo, John Rula, and Alberto Dainotti. 2021. Crowdsourced Internet Measurement: Bias and Representativeness. In *Proceedings of the Passive and Active Measurement Conference (PAM)*. Springer, 123–136. doi:10.1007/978-3-030-72582-2_8
- [38] Reinaldo Sanchez-Arias, Luis G. Jaimes, Shahram Taj, and Md. Selim Habib. 2023. Understanding the State of Broadband Connectivity: An Analysis of Speedtests and Emerging Technologies. *IEEE Access* 11 (2023), 101580–101603. doi:10.1109/ACCESS.2023.3313231
- [39] Fatih Berkay Sarpkaya, Fraida Fund, and Shivendra Panwar. 2025. To Adopt or Not to Adopt L4S-Compatible Congestion Control? Understanding Performance in a Partial L4S Deployment. In *Passive and Active Measurement*, Cecilia Testart, Roland van Rijswijk-Deij, and Burkhard Stiller (Eds.). Springer Nature Switzerland, Cham, 217–246.
- [40] James Saxon and Dan A. Black. 2022. What we can learn from selected, unmatched data: Measuring internet inequality in Chicago. *Computers, Environment and Urban Systems* 98 (2022), 101874. doi:10.1016/j.compenvurbsys.2022.101874
- [41] Koen De Schepper, Olga Albisser, Olivier Tilmans, and Bob Briscoe. 2022. Dual Queue Coupled AQM: Deployable Very Low Queuing Delay for All. arXiv:2209.01078 [cs.NI] <https://arxiv.org/abs/2209.01078>
- [42] Taveesh Sharma, Paul Schmitt, Francesco Bronzino, Nick Feamster, and Nicole P. Marwell. 2024. Beyond Data Points: Regionalizing Crowdsourced Latency Measurements. *Proc. ACM Meas. Anal. Comput. Syst.* 8, 3, Article 34 (Dec. 2024), 24 pages. doi:10.1145/3700416
- [43] M. Shreedhar and George Varghese. 1995. Efficient fair queueing using deficit round robin. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Cambridge, Massachusetts, USA) (SIGCOMM '95). Association for Computing Machinery, New York, NY, USA, 231–242. doi:10.1145/217382.217453
- [44] Jacob Smith, Srikanth Sundaresan, Will Scott, Siddharth Eravuchira, and Nick Feamster. 2022. Understanding Netflix's Fast.com: A Study of Web-Based Speed Measurement. In *Proceedings of the ACM Internet Measurement Conference (IMC)*. ACM, 458–470. doi:10.1145/3517745.3561442
- [45] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. Broadband Internet performance: A view from the edge. In *Proceedings of the Passive and Active Measurement Conference (PAM)*. Springer, 9–22. doi:10.1007/978-3-642-19260-9_2
- [46] Srikanth Sundaresan, Xiaohong Deng, Yun Feng, Danny Lee, and Amogh Dhamdhere. 2017. Challenges in inferring internet congestion using throughput measurements. In *Proceedings of the 2017 Internet Measurement Conference* (London, United Kingdom) (IMC '17). Association for Computing Machinery, New York, NY, USA, 43–56. doi:10.1145/3131365.3131382
- [47] Greg White. 2025. *Operational Guidance on Coexistence with Classic ECN during L4S Deployment*. Internet-Draft draft-ietf-tsvwg-l4sops-08. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-ietf-tsvwg-l4sops/08/> Work in Progress.

A Ethics Statement

This research was conducted in a lab environment using our own testbed infrastructure. All experiments were performed on a private network with no involvement of human subjects or measurements from external users. This work does not pose any ethical or privacy concerns.

B Code Snippets for Measurement Setup

B.1 Code Snippets for data collection with NDT7 and remote tcpdump

Listing 1: Initializing file paths and creating the remote output directory.

```
def run_speedtest_with_remote_pcap(
    output_root, server_ip):
    """
    Runs ndt7-client locally while capturing
    PCAP remotely via SSH.
    """
    current_time = time.strftime("%Y%m%d-%H%M%S")
    json_file = f"{output_root}/speedtest_{current_time}.jsonl"
    remote_pcap = f"/home/noise/{output_root}/speedtest_{current_time}.pcap"

    print(f"[*] Creating remote directory: {output_root}")
    mkdir_cmd = f"mkdir -p {output_root}"
    mkdir_result = ssh_command_server(
        mkdir_cmd)
    if mkdir_result.returncode != 0:
        print(f"[!] Failed to create remote directory {output_root}")
        return None, None

    print(f"[*] Remote PCAP will be saved to : {remote_pcap}")
```


Listing 2: Starting remote tcpdump using adapter_unsynced and background execution.

```

tcpdump_remote_cmd = (
    f"tcpdump -j adapter_unsynced -w {
        remote_pcap} host {server_ip} "
    "> /dev/null 2>&1 & echo $!"
)

print("[*] Starting remote tcpdump...")
start_proc = ssh_command_server(
    tcpdump_remote_cmd)

if start_proc.returncode != 0:
    print("[!] Failed to start remote
        tcpdump:")
    print(start_proc.stderr)
    return None, None

tcpdump_pid = start_proc.stdout.strip()
print(f"[*] Remote tcpdump started with
    PID {tcpdump_pid}")

```

Listing 3: Running the NDT7 client locally and stopping the remote capture.

```

try:
    os.system(
        f"ndt7-client -server {server_ip}
        } -no-verify "
        f"-format json > {json_file}"
    )
finally:
    print("[*] Stopping remote tcpdump
        ...")
    ssh_command_server(f"sudo kill -
        SIGINT {tcpdump_pid}")

print(f"Speedtest JSON saved locally at:
    {json_file}")
print(f"PCAP saved remotely at: {
    remote_pcap}")

return json_file, remote_pcap

```

B.2 Helper script for SSH Commands

Listing 4: Helper function to run a command via SSH on the remote server.

```

def ssh_command_server(cmd):
    key_path = "/path/.ssh/id_rsa" # adjust
        to the one from ssh -v
    server = "<name>@" + server_ip.split(":")
        [0]
    result = subprocess.run(
        ["ssh", "-o", "StrictHostKeyChecking
            =no", "-i", key_path, server,
            cmd],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True,
    )
    return result

def scp_command_server(remote_path,
    local_path):
    key_path = "/path/.ssh/id_rsa" # adjust
        to the one from ssh -v
    server = "<name>@" + server_ip.split(":")
        [0]
    result = subprocess.run(
        ["scp", "-o", "StrictHostKeyChecking
            =no", "-i", key_path, f"{server
            }:{remote_path}", local_path],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True,
    )
    return result

```

Listing 5: Helper function to run a command via SSH on the remote server.

```
def collect_remote_pcap(remote_pcap_path,
    local_output_dir):
    """
    Copies the remote PCAP file back to
    local storage.
    """
    remote_pcap_path = remote_pcap_path.
        strip()
    local_path = os.path.join(
        local_output_dir, os.path.basename(
            remote_pcap_path))

    print(f"[*] Fetching {remote_pcap_path}
        from remote host...")
    print(f"[*] Saving to local path: {
        local_path}")
    result = scp_command_server(
        remote_pcap_path, local_path)
    if result.returncode != 0:
        print("[!] Failed to fetch remote
            PCAP:")
        print(result.stderr)
        return None

    print(f" PCAP saved locally to {
        local_path}")
    return local_path
```

B.3 Code Snippet for AQM shaping with tc

Listing 6: Bash script header, variable setup, and start() function.

```
#!/bin/bash
CMD=$1
IFACE=$2
BW=$3
LATENCY=$4
LOSS=$5
AQM_METHOD=$6

TC=/sbin/tc

start() {
    if [[ -z "$BW" || -z "$LATENCY" || -z "
        $LOSS" ]]; then
        echo "Usage: $0 start <interface> <
            bandwidth_kbps> <latency_ms> <
            loss_percentage> <aqm_method>"
        exit 1
    fi

    echo "Applying traffic shaping with AQM
        ($AQM_METHOD) to $IFACE..."

    $TC qdisc del dev $IFACE root 2>/dev/
        null
    $TC qdisc add dev $IFACE root handle 1:
        htb default 11
    $TC class add dev $IFACE parent 1:
        classid 1:11 htb rate ${BW}kbit \
        ceil $((BW * 16 / 10))kbit burst $((
            BW * 36 / 1000))k \
        cburst $((BW * 36 / 1000))k
```

Listing 7: Applying the selected AQM or tail drop in start().

```
if [[ "$AQM_METHOD" == "no_aqm" ]]; then
    $TC qdisc add dev $IFACE parent 1:11
        handle 20: pfifo limit 100
    echo "Traffic shaping applied with
        tail drop (pfifo) on $IFACE"
else
    $TC qdisc add dev $IFACE parent 1:11
        handle 20: ${AQM_METHOD}
    echo "Traffic shaping applied with
        $AQM_METHOD on $IFACE"
fi
}
```

Listing 8: Stop function, show function, and case statement handling commands.

```
stop() {
    echo "Removing traffic shaping from
        $IFACE..."
    $TC qdisc del dev $IFACE root 2>/dev/
        null
}

show() {
    echo "Current traffic control settings
        for $IFACE:"
    $TC qdisc show dev $IFACE
    $TC class show dev $IFACE
    $TC filter show dev $IFACE

    AQM_CHECK=$(($TC -s qdisc show dev $IFACE
        | grep -E "fq_codel|cake|pie|sfq|
        codel")
    if [[ -n "$AQM_CHECK" ]]; then
        echo " AQM ($AQM_METHOD) is active
            on $IFACE."
    elif [[ "$AQM_METHOD" == "no_aqm" ]];
        then
        echo " Tail drop (pfifo) is applied
            instead of AQM on $IFACE."
    else
        echo " AQM is NOT applied!"
    fi
}

case "$CMD" in
    start)
        start
        show
        ;;
    stop)
        stop
        ;;
    show)
        show
        ;;
    *)
        echo "Usage: $0 {start|stop|show} <
            interface> <bandwidth_kbps> <
            latency_ms> <loss_percentage> <
            aqm_method>"
        ;;
esac
```