Steady and Transient State Analysis of CUBIC Congestion Control with FQ-CoDel and FQ-PIE

Deepa Kumari*, Suvam Mukherjee*, Vivek Jain†, Mohit P. Tahiliani*
*Wireless Information Networking Group
Department of Computer Science and Engineering, NITK Surathkal, Mangalore, Karnataka, India
{deepakumari.207cs500, suvammukherjee.217cs011, tahiliani}@nitk.edu.in
†University of California, Riverside, USA
vjain014@ucr.edu

***Abstract*—The stability of Congestion Control Algorithms (CCAs) is critical to maintaining consistent network performance, particularly as modern networks increasingly rely on Active Queue Management (AQM) and packet scheduling techniques. Although prior work has studied CCA behavior under steady-state conditions, less attention has been paid to transient states; moments of rapid change that significantly affect responsiveness, latency, and fairness. In this paper, we investigate the stability of CCAs under AQM-controlled bottlenecks, focusing on real-world scenarios with default system configurations. We evaluate the widely deployed CUBIC CCA with an emphasis on two algorithms: Flow Queue Controlled Delay (FQ-CoDel), which uses deterministic dropping, and Flow Queue Proportional Integral controller Enhanced (FQ-PIE), which uses probabilistic dropping. These differing designs reveal nuanced interactions during load shifts, highlighting trade-offs in link throughput and queue delay. The goal is to deepen the understanding of CCA-AQM dynamics under practical conditions. Our results show that FQ-PIE adapts faster in transient conditions, while both FQ-CoDel and FQ-PIE provide stronger long-term stability.**

***Index Terms*—CUBIC, FQ-CoDel, FQ-PIE**

I. Introduction

Congestion control algorithms (CCAs) are essential for maintaining the stability in both steady and transient network states. Stability refers to the ability of CCAs to promptly adapt to changing network conditions without introducing large fluctuations in throughput or delay. It is important not only for conventional Internet use, but also for modern applications such as cloud services, video streaming, IoT systems, and tactical communications, where short-lived disruptions can degrade user experience, reduce fairness, or delay time-sensitive decisions. In addition, the increasing deployment of Active Queue Management (AQM) and flow scheduling mechanisms has further introduced a new dimension that directly influences the stability of CCAs. Their interaction with CCAs can be intricate, especially during the transient state.

While existing research has evaluated the interaction of CCAs with AQM mechanisms in steady network state, it remains less thoroughly explored in transient network state. In such environments, transient behaviors – such as how quickly CCAs converge to equilibrium, how AQM mechanisms react to sudden changes in load, and how fairness evolves during these transitions – are critically important for user experience, yet often overlooked and remain unanswered. In this paper, we address these gaps by systematically analyzing the stability of CUBIC CCA [1] with two hybrid AQM and packet scheduling, FQ-CoDel [2] and FQ-PIE [3], *under both steady and transient conditions*. The significance of our work lies in its practical focus: (1) we analyze stability under both steady and transient network states, and (2) we purposefully avoid any tuning of AQM or CCA parameters that typical end-users are unlikely to perform. Hence, we have not considered performing evaluations using Explicit Congestion Notification (ECN) [4].

The contributions of this paper are: (1) systematic evaluation of CUBIC with FQ-CoDel and FQ-PIE, under both steady and transient states, with ns-3 simulations and real-world WiFi network using default parameters, and (2) identifying scenarios where FQ-CoDel outperforms FQ-PIE, and vice versa.

II. Background

Steady vs. Transient Network State: The transient state refers to periods where the network conditions are changing rapidly, such as during flow startup, link failures, or data rate fluctuations (e.g., wireless network). In contrast, the steady state is when the network's capacity allows flows to maintain a stable rate. Performance in both phases is critical: responsiveness in the transient state determines adaptation, while steady-state behavior governs fairness and throughput. These scenarios are discussed in detail in Section III.

CUBIC: It is a widely deployed CCA, especially in Linux distributions. It uses a cubic function of time since the last congestion event to determine the size of the congestion window, allowing for faster growth when the network is underutilized and probing gently when the network is operating near the estimated capacity. RFC 9438 formalizes and standardizes its behavior, addressing prior ambiguities and interoperability concerns. It introduces clarifications that improve fairness, especially when coexisting with Reno-like flows, and specifies behaviors across both steady and transient states. This paper addresses the lack of in-depth analysis of CUBIC's performance during transient states with AQM algorithms.

Bufferbloat: It refers to excessive latency caused by overly large and unmanaged buffers in network devices. When buffers are persistently full, even in the presence of congestion control, flows experience large round-trip times, degrading Quality of Experience, especially for interactive and real-time traffic.

FQ-CoDel and FQ-PIE: These are queue management algorithms designed to combat bufferbloat by integrating AQM

978-3-903176-75-1 ©2025 IFIP

mechanisms with per-flow scheduling. Both use Deficit Round Robin (DRR) schedulers to isolate flows and prevent queue monopolization, and they apply AQM techniques, CoDel or PIE, to manage queue delay dynamically.

Our rationale for protocol selection is as follows: We choose CUBIC because it is the default CCA in most major operating systems (e.g., Linux, macOS). For AQM, we focus on FQ-CoDel and FQ-PIE, which represent contrasting design philosophies: deterministic packet dropping in FQ-CoDel and probabilistic dropping in FQ-PIE. These complementary approaches make them ideal candidates for comparison.

III. Methodology and Measurement Setup

This section describes our methodology which consists of different scenarios, network topologies and tools we used. Our topologies and scenarios are selected to reflect typical real-world deployments, ensuring practical relevance of this work.

A. Steady and Transient Network Scenarios

Our evaluation is based on the recommendations provided in RFC 9743, which provides a framework for developing and assessing CCAs. This work mainly focuses on the evaluation criteria specified for evaluating single algorithm behavior, specifically, Protection against Bufferbloat as described in Section 5.1.2 and transient events as described in Section 7.8 of RFC 9743. Our evaluation scenarios can be classified into two main categories: (i) steady network state and (ii) transient network state. In every scenario, we assess the stability of CCA in the presence of different AQM mechanisms.

1) Steady-state experiments: In steady network state, our goal is to stress-test (e.g., presence of staggered multiple flows) the aspect of CCAs even when the network conditions are not varying. Specifically, we conduct three types of experiments to evaluate the behavior of the CCA under steady network conditions: (a) Synchronous Fairness experiment: Multiple flows starting at the same time under identical network conditions; (b) Staggered Start Time Fairness experiment: Multiple flows experiencing identical network conditions but starting at different times, and (c) RTT Fairness experiment: Multiple flows with different RTTs experiencing identical network conditions. Scenarios (b) and (c) represent real-world network conditions, where flows start at different times and have varying RTTs.

2) Transient-state experiments: We perform two experiments for analyzing the behavior of CCA in a transient network state: (a) Step function: the value of one of the network parameters is changed once in an experiment. This experiment mimics a sudden change in the network conditions, for e.g., a sudden drop in the available bandwidth for a connection. (b) Pulse wave: the value of one of the network parameters oscillates between two values periodically; for e.g., the available bandwidth of a connection rises and falls.

B. Network Topologies

We use two network topologies for this work: (i) a standard dumbbell topology with 10 TCP sender and receiver pairs as shown in Fig. 1. The traffic is unidirectional. For all the

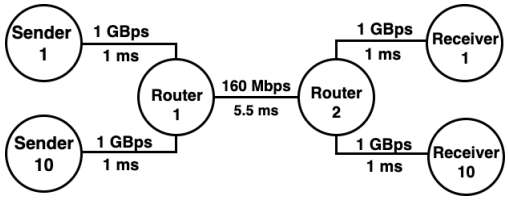


Fig. 1. Dumbbell topology with 10 TCP sender and receiver pairs

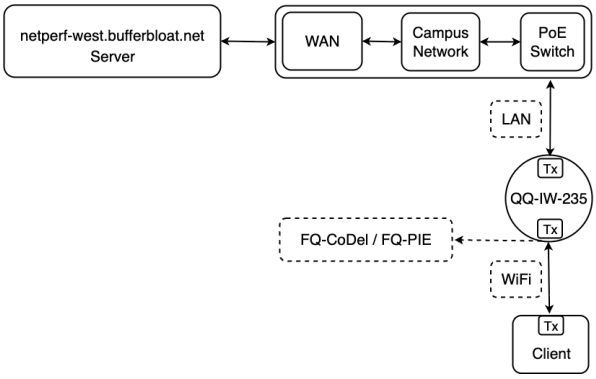


Fig. 2. Network topology with QQ-IW-235 WiFi Access Point

experiments using this topology, we have used 160 Mbps as the bottleneck capacity, 15 ms as the RTT, and 1000 packets as the queue size. All edge links are 1GBps. (ii) a real world WiFi network as shown in Fig. 2. The last hop connecting the laptop and WiFi access point acts as a bottleneck.

All senders use CUBIC, and FQ-CoDel/FQ-PIE are used at the bottleneck link. None of the default parameters are changed in CUBIC, FQ-CoDel, and FQ-PIE. We consider the aggregated throughput of all flows passing through a bottleneck router and the queue delay experienced by packets on that router as the evaluation metrics for all experiments.

C. Evaluation Tools

We use ccperf [5], built on top of ns-3 [6], as one of the tools to simulate the topology shown in Fig. 1 due to its automated, reproducible framework for benchmarking CCAs. It supports fine-grained simulation control, built-in metric collection, and extensible experiment design. Its modular, declarative setup and support for fairness and stability analysis make it suitable for this work. Besides, the simulation scenarios provided in ccperf closely match those recommended in RFC 9743.

We use Flexible Network Tester (flent) [8], specifically the Realtime Response Under Load (RRUL) and *tcp_n* download tests to run experiments on a real topology shown in Fig. 2. QQ-IW-235 WiFi access point of Quantum Networks has been used for the evaluation. It uses OpenWRT 23, Linux kernel version is 5.15.153 and iproute2 version is 6.3.0. FQ-CoDel/FQ-PIE are enabled on QQ-IW-235.

IV. Measurement Results

A. Steady State Analysis using ccperf

Figures 3 and 4 show the throughput and queue delay achieved with FQ-CoDel and FQ-PIE in all steady state scenarios explained in Section III. We show the results with

FIFO to highlight the significance of using AQM mechanisms, but do not discuss its results in detail. The primary focus is on the performance obtained by using FQ-CoDel or FQ-PIE. For the experiments related to RTT Fairness, ten flows have the following RTT: 10ms, 19ms, 24ms, 33ms, 41ms, 49ms, 55ms, 62ms, 64ms and 81ms, respectively. For the experiments related to Start Time Fairness, ten flows are initiated at staggered times: 0s, 910ms, 920ms, 1.06s, 1.46s, 1.52s, 2.16s, 2.62s, 3.06s, and 4.16s, respectively.

In Fig. 3a, we observe that FIFO and FQ-PIE fully utilize the network capacity, whereas throughput drops sporadically with FQ-CoDel. On deeper analysis, it was observed these drops in throughput are due to the working of control law in CoDel (and hence, FQ-CoDel). The control law in CoDel reacts slowly when it enters the dropping phase, but eventually drops aggressively if the queue delay does not remain with the target delay of 5ms. These aggressive packet drops lead to an empty queue, leading to loss of throughput. This can be confirmed by noting that queue delay in FQ-CoDel drops to 0 in Fig. 4a. Fig. 3c shows that both algorithms stabilize instantly after the last flow among the ten joins the network (at 4.16s). The other plots show that FQ-CoDel and FQ-PIE perform as expected, where FQ-CoDel and FQ-PIE attempt to maintain the target queue delay to 5ms and 15ms, respectively.

Fig. 5 provides an in-depth information about per flow throughput received when flows with different RTT and start times, respectively. It shows that the AQM algorithms help CCA achieve bandwidth utilization when flows with different RTTs exist. CUBIC TCP shows strong RTT unfairness under FIFO, where long RTT flows are penalized with much lower throughput as shown in Fig. 5a. For instance, the flow with RTT 41ms gains 52.60Mbps, whereas the flow with RTT 64ms achieves 6.58Mbps. We also observe that the throughput distribution shows significant variation based on RTTs. FQ-CoDel and FQ-PIE significantly mitigate this unfairness, achieving near-equal throughput across all flows regardless of RTT. The result obtained confirms that queuing discipline is decisive in how fairly CUBIC handles RTT heterogeneity.

Table I shows the fairness calculated for the steady state scenarios using the HARM index [7]. It examines if a flow harms another flow when competing for resources. A value near zero indicates *harmless* and a value near one represents *harmful*. The incentive to deploy AQM is evident from the HARM range obtained for FIFO. The throughput is uneven among ten flows and queue delay is high. The HARM ranges for FQ-CoDel and FQ-PIE are similar for Synchronous and Start Time Fairness scenarios, all being very close to 0. FQ-PIE performs slightly better in terms of throughput and queue delay in RTT Fairness scenario. The flow with a RTT of 81ms gets lower throughput than its share when FQ-CoDel is used, leading to slightly higher HARM values. Nonetheless, FQ-CoDel and FQ-PIE successfully control the queue delay.

B. Transient State Analysis using ccperf

We perform two sets of experiments for analyzing the behavior of CUBIC in a transient network state.

1) Step Response: Four separate scenarios are simulated: increasing bottleneck data rate from 16Mbps to 64Mbps once, decreasing bottleneck data rate from 64Mbps to 16Mbps once, increasing RTT from 15ms to 30ms once, and decreasing RTT from 30ms to 15ms once. Figures 6 and 7 show the results.

Data Rate up from 16Mbps to 64Mbps: At 5s, the data rate is increased. The goal is to evaluate how well the CUBIC TCP+AQM setup adapts to a sudden increase in available capacity. The main questions driving this analysis are: (a) Does throughput ramp up efficiently to utilize the available capacity? (b) How do different AQM algorithms handle queuing delays during the transition? (c) Is there an initial overshoot or under utilization due to slow adaptation?

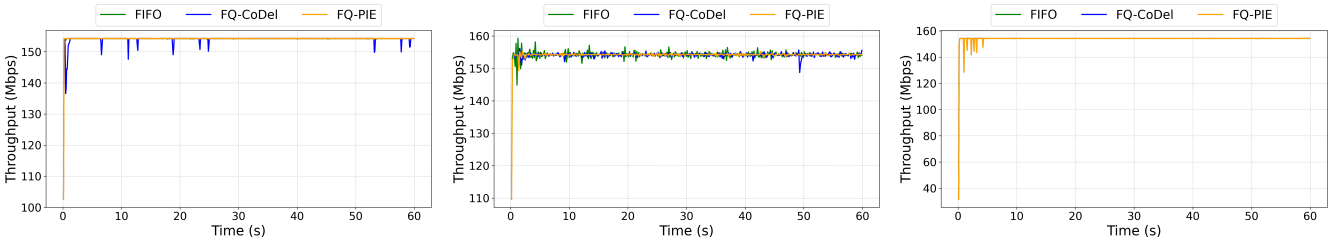
Queue delay increases with FQ-PIE at the beginning as shown in Fig. 6c, resulting in a slight throughput dip at the start of the simulation, seen in Fig. 6a. FQ-PIE controls the queue delay thereafter, stabilizing the network throughput. When the Bandwidth Delay Product (BDP) of the network increases at 5s, FQ-CoDel drains off all the packets quickly, leading to poor network utilization. As we observe in Fig. 6a, FQ-CoDel takes nearly 5 seconds to utilize the available capacity, whereas FQ-PIE and FIFO do it much faster. When the data rate increases from 16 Mbps to 64 Mbps, FQ-PIE allows small burst transmission without dropping the packet and can quickly grab the available network capacity. In FQ-CoDel, a small burst may increase the target queue delay, resulting in slower growth in achieving the available bandwidth. Nevertheless, both FQ-CoDel and FQ-PIE maintain acceptable queue delays.

Data Rate down from 64Mbps to 16Mbps: At 5s the data rate is decreased. The goal is to observe how the system reacts to a sudden drop in available data rate. The questions driving this analysis are: (a) Does throughput stabilize at lower data rate quickly, or is there a prolonged instability? (b) Does the queue delay spike due to the continuous high sending rate?

When data rate drops from 64Mbps to 16Mbps, all algorithms achieve a stable throughput, as shown in Fig. 6b. FQ-Codel and FQ-PIE successfully avoid the queue delay spike when data rate drops and keep the queue delay under the target, as shown in Fig. 6d. Queue delay spikes with FIFO at 5s.

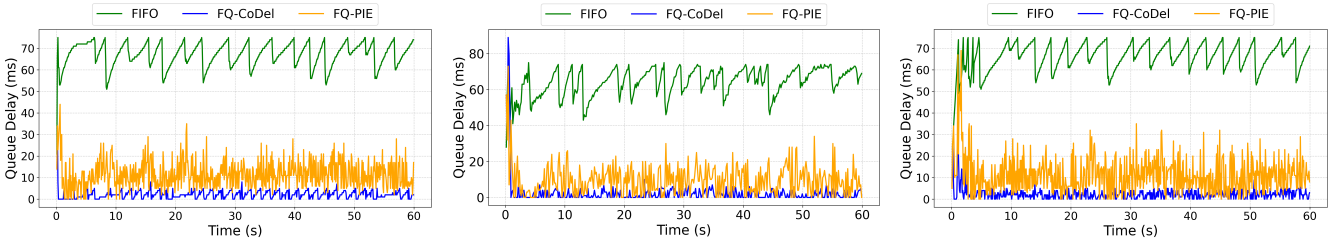
RTT up 15ms to 30ms: At 5s the network RTT increases from 15ms to 30ms. The goal of this experiment is to evaluate the responsiveness of CUBIC to increasing network delay. The main questions driving this analysis are: (a) How does the sudden RTT increase impact throughput across AQM algorithms? (b) Does any AQM algorithm adapt better to keep the delay under control during higher RTT?

We observe a throughput dip of 18% with FQ-CoDel at 5s in Fig. 7a. RTT increase leads to BDP increase, requiring more in-flight data to fully utilize the link and avoid under utilization of the available data rate. FQ-CoDel maintains a relatively low queue and hence drains it quickly when the BDP increases, leading to significant loss of throughput. CUBIC's congestion window (*cwnd*) growth becomes slower with an increase in RTT; hence, the sender does not immediately scale the window to match the new capacity, leading to low throughput. Fig. 7c confirms this behavior where FQ-CoDel often results in



(a) Synchronous Fairness (b) RTT Fairness (c) Start Time Fairness

Fig. 3. Throughput for Steady State Scenarios

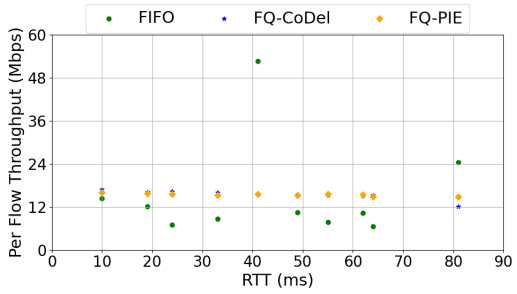


(a) Synchronous Fairness (b) RTT Fairness (c) Start Time Fairness

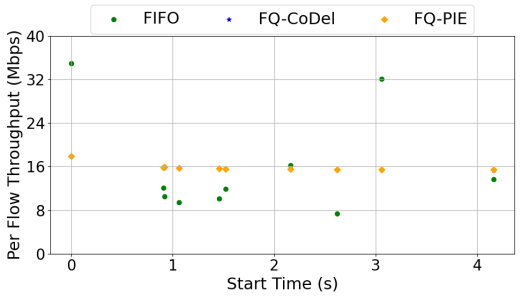
Fig. 4. Queue Delay for Steady State Scenarios

TABLE I
HARM METRICS COMPARISON FOR DIFFERENT AQM MECHANISMS UNDER STEADY STATE ANALYSIS

AQM	HARM Metric	Flow IDs	1 Flow	Synchronous Fairness		Start Time Fairness		RTT Fairness	
				10 Flows	HARM Range	10 Flows	HARM Range	10 Flows	HARM Range
FIFO	Throughput	1–10	15.42	9.6–25.34	-0.644 to 0.377	7.37–34.96	-1.268 to 0.522	6.58–52.60	-2.412 to 0.573
	Queue Delay (ms)	–	578.39	67.17	-0.884	66.14	-0.886	64.28	-0.889
FQ-CoDel	Throughput	1–10	15.38	15.37	∼0.0	15.38–15.79	-0.032 to -0.000	12.17–16.84	-0.095 to 0.209
	Queue Delay (ms)	–	2.15	2.12	-0.016	2.41	0.121	2.46	0.144
FQ-PIE	Throughput	1–10	15.42	15.34–15.43	-0.001 to 0.005	15.43–15.90	-0.031 to -0.001	14.85–15.96	-0.035 to 0.037
	Queue Delay (ms)	–	11.63	11.64	0.001	11.38	-0.021	9.67	-0.168



(a) RTT Fairness



(b) Start Time Fairness

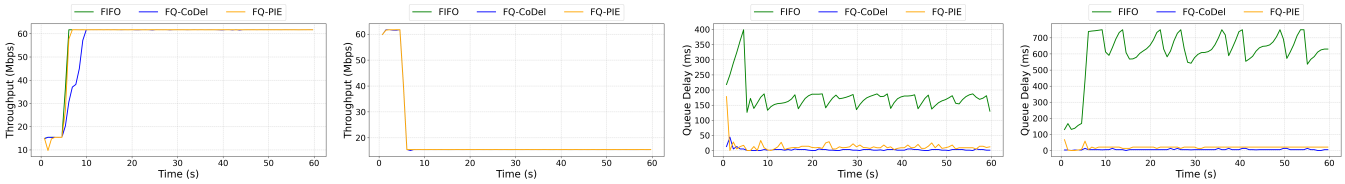
Fig. 5. Per Flow Throughput for Steady State Scenarios

near-zero queue depths, maintaining fewer packets in-flight than needed, and leads to poor link utilization frequently. The throughput dips in case of FQ-PIE when it encounters the sudden increase in RTT; however, the PI controller manages to keep the network stabilized subsequently, and the queue delay is controlled within the target.

RTT down from 30ms to 15ms: At 5s the network RTT is decreased. The goal of this experiment is to evaluate responsiveness in low RTT conditions. The main question driving this analysis is: which AQM algorithm best leverages the low RTT environments for improved latency and throughput?

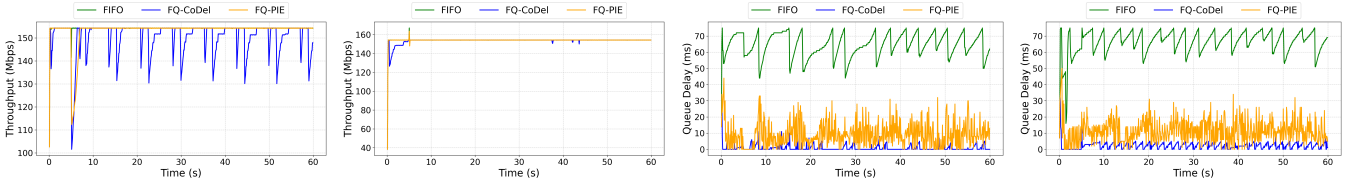
As shown in Fig. 7b, when the RTT is initially high, FQ-CoDel takes some time to fully utilize the link. *cwnd* increments are slower when RTT is more, which leads to sudden bursts of packets being sent by ten CUBIC senders. This leads to a sudden increase in the queue delay for all algorithms, as shown in Fig. 7d. FQ-CoDel aggressively drops packets and ends up getting a zero queue delay, resulting in loss of throughput seen in Fig. 7b for the first 5 seconds. Nonetheless, the state improves subsequently and remains stable throughout. The performance of FQ-PIE is as expected.

2) Response to Pulse Wave: Two separate scenarios are simulated in this experiment: periodically vary the data rate



(a) Data Rate Increase: Throughput (b) Data Rate Decrease: Throughput (c) Data Rate Increase: Queue Delay (d) Data Rate Decrease: Queue Delay

Fig. 6. Step Response to Data Rate Increase (16 Mbps to 64 Mbps) and Data Rate Decrease (64 Mpbs to 16 Mbps)



(a) RTT Increase: Throughput (b) RTT Decrease: Throughput (c) RTT Increase: Queue Delay (d) RTT Decrease: Queue Delay

Fig. 7. Step Response to RTT Increase (15 ms to 30 ms) and RTT Decrease (30 ms to 15 ms)

from 16Mbps to 64Mbps to 16Mbps, and RTT from 15ms to 30ms to 15ms every 5 seconds. When the data rate increases, it opens up additional resources that the CCA can grab by increasing the sending rate. When the data rate reduces, congestion arises and packets drop, forcing the CCA to reduce its sending rate. The periodic changes in the data rate impact the CCA's sending rate. On the other hand, sudden changes in the RTT also impact the sending rate of CUBIC, as the window growth function is proportional to the RTT. The main goal of these experiments is to check if CCA's sending rate follows the pulse wave function, and confirm whether the AQM algorithms play an essential role in guiding the CCA about the network condition. Fig. 8 shows the results along these lines.

Varying Data Rate: When the data rate is varied from 16Mbps to 64Mbps every 5 seconds, we observe that FIFO follows the pulse wave function by fully utilizing the network bandwidth (Fig. 8a) but at the cost of high queuing delay (Fig. 8c). The PI controller helps FQ-PIE to catch up with the available network capacity before it encounters the next drop in the data rate. Meanwhile, FQ-CoDel is unable to utilize the additional bandwidth available before it drops again. This is inline with our observations from Fig. 6a where FQ-CoDel needs time to catch up with additional bandwidth because it maintains a relatively lower queue length. Both FQ-CoDel and FQ-PIE successfully maintain lower queue delays.

Varying RTT: We observe that when RTT increases, the throughput drops significantly with FQ-CoDel, as shown in Fig. 8b. It is mainly because it does not have sufficient number of packets in the queue (Fig. 8d) to keep the BDP full. FQ-PIE manages both throughput and queue delay appropriately. The error correction in PI controller helps FQ-PIE to adapt to dynamically changing network conditions.

C. Steady State Analysis using QQ-IW-235

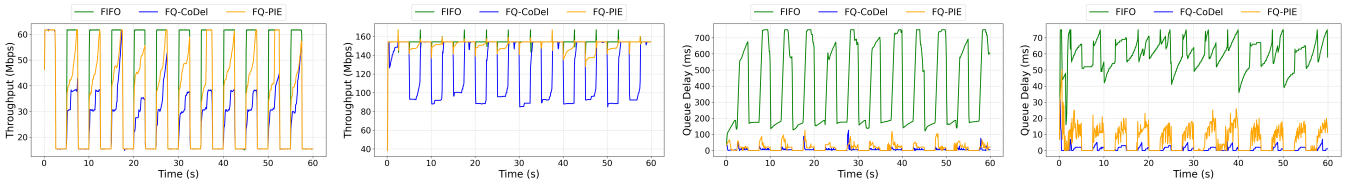
Figures 9 and 10 show the results obtained for FQ-CoDel and FQ-PIE from RRUL and *tcp_n* download test conducted on a live WiFi network using QQ-IW-235 access points, respectively. RRUL is one of the popular test suite to analyze the network performance under the heavy workloads that typically induce bufferbloat. It loads up the link with eight TCP streams (four downloads, four uploads), against ICMP ping (for RTT measurements) and UDP traffic. *tcp_n* download test allows us to test the performance with varying number of TCP flows emulating heavy download traffic.

The box in Figures 9 and 10 represents Interquartile Range (IQR) covering the 25th to 75th percentiles. The median (50th Percentile) is shown as a line within the box. The whiskers extend up to 1.5 times the IQR, indicating the expected range of data. Data points outside the whiskers are considered outliers. The ellipsis plot in Figures 9 and 10 provides a visual representation of the relationship between throughput (Y-axis) and latency in terms of RTT (X-axis). The ellipse represents a region covering approximately 86.5% of the data points, assuming a normal distribution. The median is marked too. The experiment duration in both the tests is 60 seconds, and all results have been averaged over 25 runs.

Results from RRUL test shows that FQ-PIE has lesser variations in queue delay, and those from *tcp_n* download show that both FQ-CoDel and FQ-PIE have similar performance.

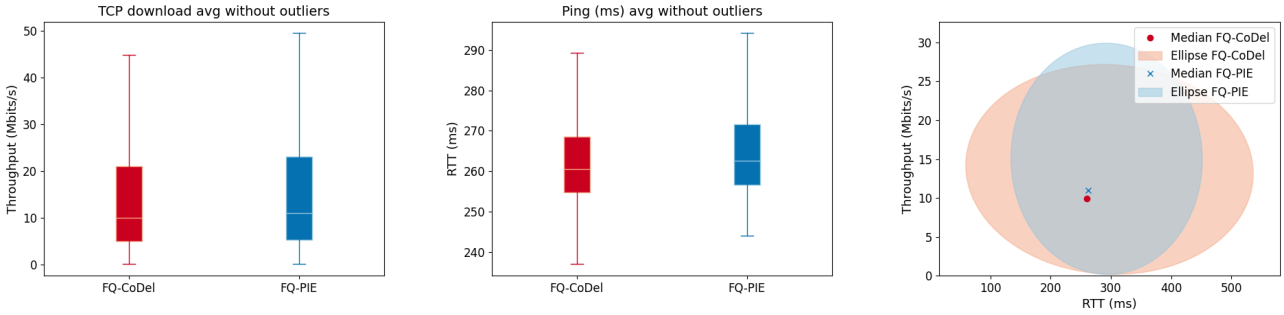
D. Inferences

Our results from ns-3 simulations and live network tests conducted over a variety of dynamic network conditions confirm that AQM algorithms help CUBIC to adapt to varying network conditions and achieve network stability in terms of throughput and queue delay. Although FQ-CoDel and FQ-PIE offer similar benefits, FQ-CoDel requires attention when a bulk of additional data rate becomes available (although CUBIC natively has a feature to capture any additional bandwidth, when available), and when RTT in the network varies. It is slow in adapting to these network situations, both of which are common in WiFi networks. The rate adaptation algorithms in WiFi may lead to frequent changes in the data rate, and the portable nature of devices can lead route changes, and consequently variations in RTT. There are two potential points that require a detailed evaluation: (i) should CoDel/FQ-CoDel



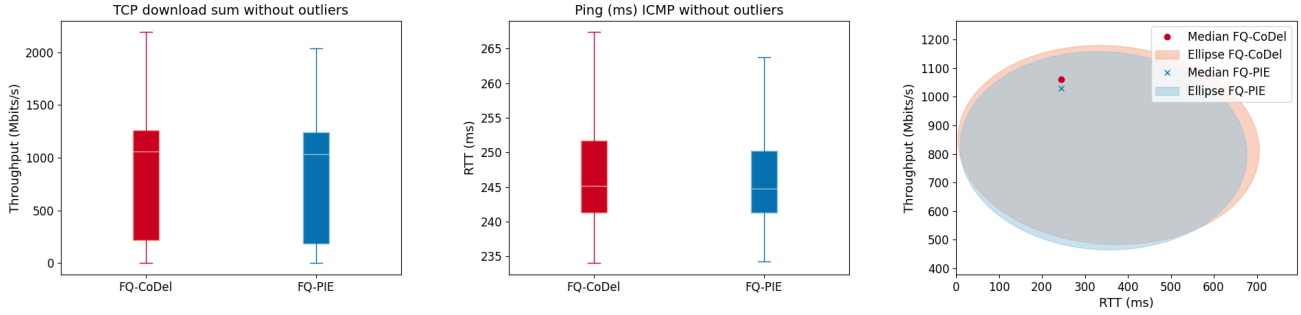
(a) Varying Data Rate: Throughput (b) Varying RTT: Throughput (c) Varying Data Rate: Queue Delay (d) Varying RTT: Queue Delay

Fig. 8. Pulse Response to Data Rate and RTT variations



(a) TCP download throughput (average) (b) Ping RTT (ms) (c) Ellipsis

Fig. 9. FQ-CoDel vs FQ-PIE using RRUL test



(a) TCP download throughput (sum): 200 flows (b) Ping RTT (ms): 200 flows (c) Ellipsis: 200 flows

Fig. 10. FQ-CoDel vs FQ-PIE using *tcp_n* download test

use a value higher than 5ms as its target queue delay? (ii) is CoDel's control law suitable for CUBIC like flows that are typically more aggressive than traditional Reno flows?

V. Conclusions and Future Work

This work investigates the stability of CUBIC CCA to maintain consistent network performance, particularly where the network relies on AQM and packet scheduling techniques like FQ-CoDel and FQ-PIE. Stability of CCA is analyzed in terms of throughput and queue delay under both steady state and transient state conditions, using simulations and real network deployed in a campus network. We derived inferences that can help further investigations to improve the performance of CUBIC with FQ-CoDel, both of which are deployed widely, specially in cases when the available data rates vary or the RTT of the path varies.

Acknowledgment

The authors thank Zen Exim Pvt. Ltd. (Quantum Networks) for their generous support in providing QQ-IW-235 WiFi Access Points. This work was also funded by Zen Exim Pvt. Ltd. (Quantum Networks).

References

[1] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast and Long-Distance Networks," RFC 9438, Internet Engineering Task Force (IETF), Aug. 2023.

[2] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," RFC 8290, Internet Engineering Task Force (IETF), Dec. 2018.

[3] G. Ramakrishnan, M. Bhasi, V. Saicharan, L. Monis, S. D. Patil, and M. P. Tahiliani, "FQ-PIE Queue Discipline in the Linux kernel: Design, Implementation and Challenges," In Proceedings of 44th IEEE LCN Symposium on Emerging Topics in Networking (LCN Symposium), 2019, pp. 117–124.

[4] K. K. Ramakrishnan, S. Floyd and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, Internet Engineering Task Force (IETF), 2001.

[5] ccperf, "ccperf: Congestion Control Performance," [Online]. Available: https://ccperf.net/

[6] ns-3 Project, "ns-3: Discrete-Event Network Simulator for Internet Systems," [Online]. Available: https://www.nsnam.org/

[7] R. Ware, Matthew K. Mukerjee, S. Seshan, and J. Sherry, "Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms," In Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19). ACM, pp. 17–24, 2019.

[8] T. Høiland-Jørgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The Flexible Network Tester," In Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools, pp. 120-125, 2017.