electronics

MDPI

Article

TCP Congestion Control Algorithm Using Queueing Theory-Based Optimality Equation

Dumisa Wellington Ngwenya [1,2,*], Mduduzi Comfort Hlophe [1] and Bodhaswar T. Maharaj [1]

[1] Department of Electrical, Electronic and Computer Engineering, University of Pretoria, Pretoria 0002, South Africa; u16250444@tuks.co.za (M.C.H.); sunil.maharaj@up.ac.za (B.T.M.)
[2] Research and Innovation, SENTECH SOC Limited, Johannesburg 2040, South Africa
* Correspondence: ngwenyad@sentech.co.za

Abstract: Internet congestion control focuses on balancing effective network utilization with the avoidance of congestion. When bottleneck bandwidth and network buffer capacities are exceeded, congestion typically manifests as packet loss. Additionally, when packets remain in buffers for too long, a queueing delay occurs. Most existing congestion control algorithms aim to solve this as a constraint satisfaction problem, where constraints are defined by bandwidth or queueing delay limits. However, these approaches often emphasize finding feasible solutions over optimal ones, which often lead to under-utilization of available bandwidth. To address this limitation, this article leverages Little's Law to derive a closed-form optimality equation for congestion control. This optimality equation serves as the foundation for developing a new algorithm, TCP QtColFair, designed to optimize the sending rate. TCP QtColFair is evaluated against two widely deployed congestion control algorithms: TCP CUBIC, which utilizes a cubic window growth function to enhance performance in high-bandwidth, long-distance networks and TCP BBR (Bottleneck Bandwidth and Round-trip propagation time), developed by Google to optimize data transmission by estimating the network's bottleneck bandwidth and round-trip time. In terms of avoiding queueing delays and minimizing packet loss, TCP QtColFair outperforms TCP CUBIC and matches TCP BBR's performance when network buffers are large. For effective network utilization, TCP QtColFair outperforms both TCP BBR and TCP CUBIC. TCP QtColFair achieves an effective utilization of approximately 96%, compared to just above 94% for TCP BBR and around 93% for TCP CUBIC.

Keywords: TCP/IP; congestion control; Kleinrock's principle; Stidham's optimality; Little's law; network optimization; queueing theory; TCP CUBIC; TCP BBR.

Academic Editors: Imtiaz Mahmud, Shakil Ahmed and Bashir Mohammed

Received: 4 November 2024
Revised: 1 January 2025
Accepted: 7 January 2025
Published: 10 January 2025

Citation: Ngwenya, D.W.; Hlophe, M.C.; Maharaj, B.T. TCP Congestion Control Algorithm Using Queueing Theory-Based Optimality Equation. *Electronics* **2025**, *14*, 263. https://doi.org/10.3390/electronics14020263

1. Introduction

For decades, end-to-end congestion control algorithms (CCAs) have been the dominant approach in Transmission Control Protocol (TCP) congestion control, despite ongoing criticisms and challenges [1,2]. These algorithms continue to be favored over hop-by-hop or network-supported mechanisms due to their compatibility with the end-to-end design principle, which promotes simplicity and effectiveness across diverse networks [3–6]. It is highlighted in [3] that network-based approaches are often avoided because they introduce per-connection control complexity within the network infrastructure. Furthermore, in [4] it is noted that network support is most effective when integrated with end-to-end CCAs, underscoring the reliance on end-to-end methods even when network-based enhancements are employed. Additionally, the observation in [7] is that mechanisms like Explicit Con-

gestion Notification (ECN) can perform poorly in environments with multiple bottlenecks, further reinforcing the preference for end-to-end approaches.

The widespread adoption of end-to-end CCAs is supported by extensive research, standardization efforts, and real-world deployments. For instance, a study in [8] found that the top 20,000 global websites predominantly rely on end-to-end TCP CCAs. A notable milestone is the standardization of TCP CUBIC [9,10] by the Internet Engineering Task Force (IETF) [11], reflecting years of iterative improvements, research, and practical implementations. Recent studies continue to highlight the relevance of end-to-end CCAs for emerging use cases such as 5G networks and data centers [7,12]. Even newer congestion control algorithms leveraging Machine Learning (ML), as demonstrated in [13–15], operate within the end-to-end paradigm. The enduring preference for end-to-end TCP CCAs is attributed to their inherent scalability, simplicity, ease of implementation, robustness, and the historical momentum they have gained through widespread use [3,4,6].

However, these algorithms face notable challenges due to the decentralized and distributed nature of the Internet [6]. In such environments, senders have limited visibility into real-time network conditions and lack direct coordination with competing flows. Consequently, end-to-end CCAs must infer congestion and available bandwidth implicitly, often leading to delayed and inaccurate congestion signals, sub-optimal responses, and stability issues. Despite numerous enhancements, many end-to-end algorithms still fall short of achieving optimal performance [16,17], highlighting the need for ongoing research and the exploration of alternative approaches [18].

1.1. Issues

Congestion typically manifests in two forms: packet loss or queueing delay. Packet loss occurs when bottleneck bandwidth (BtlBW) and network buffer capacity are exceeded, causing packets to be dropped. Queueing delay arises when packets accumulate in network buffers, resulting in longer waiting times. Therefore, most end-to-end TCP congestion control mechanisms rely on either packet loss (Loss-Based algorithms) or queueing delay (Delay-Based algorithms) as congestion indicators. Loss-Based algorithms react to congestion after it occurs, responding only when packet loss is detected, unless there is network-support [4]. In networks with large buffers, Loss-Based algorithms are prone to buffer bloat, high queueing delays and packet-loss-ratio (PLR) [2,19].

In contrast, Delay-Based algorithms aim to detect and mitigate congestion proactively by responding early to signs of growing queue lengths [17,20]. Despite this advantage, Delay-Based algorithms suffer from measurement errors, detection delays, and model inaccuracies [2,18,21,22].

A specific subclass of Delay-Based algorithms is Rate-Based algorithms, which directly compute the sending rate based on measurements of propagation delay (PropDelay) and available bandwidth estimates. However, like other Delay-Based approaches, they are vulnerable to measurement inaccuracies and detection delays, leading to over-utilization or under-utilization. For example, challenges discussed in [16,23] for TCP Bottleneck Bandwidth and Round-trip propagation time (BBR) [24] include bias against shorter round-trip time (RTT) and degradation when RTT variability is high.

Another issue with existing algorithms is their reliance on successive constraint satisfaction (SCS) heuristics to adjust sending rates in response to congestion. These heuristics, often based more on intuition than formal mathematical rationale, focus on finding feasible solutions rather than optimal ones. In Loss-Based algorithms, SCS heuristics can lead to high-amplitude oscillations, reducing throughput and network utilization [2]. Although Delay-Based algorithms integrate mathematical models for optimality [17,20], they still exhibit oscillatory behavior. Additionally, due to their sensitivity to network measurements,

they often display abrupt and jerky adjustments, particularly under dynamic network conditions [7,22].

Empirical studies in [12,25] have demonstrated that even widely deployed algorithms, such as TCP CUBIC and TCP Bottleneck Bandwidth and Round-trip propagation time (BBR), are not immune to significant oscillations. While these algorithms generally achieve high network utilization, their pronounced oscillatory behavior can degrade overall performance, especially in environments with fluctuating traffic or variable RTTs.

1.2. Contribution

This article introduces a novel Delay-Based congestion control approach based on Little's Law [26]. The main contributions are as follows:

- A novel Delay-Based congestion control approach grounded in queueing theory and Little's Law.
- Development and implementation of an algorithm based on the proposed approach.
- Performance evaluation and comparison with widely used algorithms, TCP CUBIC as in [9] and TCP BBR version 1 [24] (both as implemented in ns-3.41).

The proposed approach avoids reliance on heuristic methods by continuously solving a closed-form optimality equation derived from Little's Law [26,27]. This equation takes the form of a differential equation, capturing the rate of change in delay and data-in-flight with respect to the sending rate. By using this predictive approach, the algorithm mitigates oscillations and improves steady-state performance.

A notable advantage of this approach is that it eliminates the need for direct bandwidth measurements. Instead, the algorithm operates by setting a target RTT and adjusting the sending rate using the derived optimality equation. To the best of the authors' knowledge, this approach is novel, with no prior comparable work beyond preliminary discussions in [27].

1.3. Implementation and Evaluation

The article presents two implementations of the proposed mechanism:

- Basic Implementation: Demonstrates the fundamental convergence and adaptability of the proposed mechanism in an ideal static network environment.
- Practical Implementation: Accounts for common network dynamics, such as varying available bandwidth, PropDelay, multiple competing flows, and fairness requirements.

Both implementations are evaluated under static and dynamic network scenarios and compared with TCP CUBIC [9,10] and TCP BBR [24]. The results highlight the ability of the proposed mechanism to reduce queueing delay, prevent packet loss, and maximize network utilization.

1.4. Article Structure

The remainder of the article is structured as follows:

- Section 2 reviews the evolution of TCP CCAs and explores relevant literature on queueing theory-based optimality principles.
- Section 3 provides a background on TCP congestion control using a functional block diagram and introduces a queueing model for Transmission Control Protoco/Internet Protocol (TCP/IP) networks.
- Section 4 develops the closed-form optimality equation and describes the implementation of the proposed congestion control mechanism.
- Section 5 presents performance evaluations of the proposed algorithm compared to TCP CUBIC and TCP BBR.
- Section 6 summarizes the findings and concludes the article.

A list of abbreviations and symbols used throughout the article is provided in a designated section before the appendices.

2. Related Work

2.1. Threads on End-to-End TCP CCAs

A comprehensive examination of the evolution and classification of end-to-end TCP CCAs is presented in several key studies [18,28–30]. These works explore the progression of TCP congestion control in both wired and wireless networks, highlighting ongoing efforts to enhance performance, reliability, and adaptability. Works in [28,29] focus on preserving TCP's host-to-host architecture, offering insights into how foundational principles can be maintained while achieving performance improvements. Meanwhile, surveys in [18,30] identify pressing research challenges, particularly the need for congestion control algorithms that can adapt to dynamic and heterogeneous network environments. These studies also suggest potential directions for future research, especially in the context of emerging technologies like 5G and beyond.

The existing literature categorizes TCP CCAs into three primary types: Loss-Based, Delay-Based, and Hybrid algorithms.

- Loss-Based Algorithms are reactive, responding to congestion only after it manifests as packet loss.
- Delay-Based Algorithms are proactive, aiming to detect congestion early by monitoring queue growth.
- Hybrid Algorithms (also known as Loss-Delay-Based) primarily rely on packet loss as the congestion trigger but use delay information to fine-tune rate adjustments. Delay-Based algorithms augmented with Loss-Based techniques, however, are typically not classified as hybrids.

Despite their different approaches, both Loss-Based and Delay-Based algorithms suffer from a common limitation: they struggle to eliminate oscillations and achieve steady-state accuracy under dynamic network conditions [1,12].

As noted in [8,31], there are currently two widely deployed CCAs, TCP CUBIC [9,10], which is a Loss-Based algorithm, and TCP BBR [24], which is a Delay-Based algorithm. These two algorithms serve as benchmarks for modern high-speed networks and are frequently used as comparative references in studies such as in [12]. Their widespread adoption reflects the balance between throughput optimization (TCP CUBIC) and latency minimization (TCP BBR), making them critical points of reference for ongoing research in congestion control.

2.1.1. Evolution of Loss-Based Algorithms

The origins of Loss-Based algorithms trace back to Jacobson's seminal work [32], which introduced the Additive-Increase, Multiplicative-Decrease (AIMD) principle to prevent Internet congestion collapse. This principle underpins TCP Reno and its successor, TCP NewReno [30,33,34]. TCP NewReno's conservative rate adjustments, however, limit performance in high-speed, high-latency networks [28,29].

To address these limitations, TCP CUBIC [9] was developed, featuring a cubic growth function to optimize performance over high Bandwidth-Delay-Product (BDP) networks. While TCP CUBIC improves throughput, it remains prone to queueing delays and packet losses, leading to high-amplitude oscillations [12] that reduce goodput and effective utilization. These shortcomings have renewed interest in Delay-Based approaches, which aim to minimize queuing delays and packet loss.

2.1.2. Delay-Based Algorithms

TCP BBR [24], as a prominent Delay-Based algorithm, addresses issues inherent in Loss-Based algorithms, such as bufferbloat and high latency. By limiting in-flight data to approximately 2 × BDP, TCP BBR balances throughput and latency, inspired by Kleinrock's optimality principle [17]. Studies, such as in [24], show that TCP BBR often outperforms TCP CUBIC in terms of both throughput and latency.

However, TCP BBR is not without challenges of overshooting and oscillations. It has been shown in [12,16] that the causes of TCP BBR's challenges stem from the bandwidth and RTT probing mechanisms.

Other recently developed Delay-Based algorithms, such as TCP Copa [35], clearly set an RTT target to be met based on minimum RTT (minRTT) for finer rate control. Yet, like BBR, they struggle with precise bandwidth and delay measurements, impacting their real-world performance.

2.1.3. Data-Driven Machine Learning Congestion Control

Recently, data-driven ML approaches have emerged as promising tools for TCP congestion control [36,37]. These methods leverage historical data and adaptive learning to dynamically optimize congestion control mechanisms. For instance, in [13] Deep Reinforcement Learning (DRL) is used to dynamically select the most effective CCA based on real-time performance evaluations. Similarly, Aurora in [38] maps historical traffic patterns to determine optimal sending rates. These methods showcase ML's potential to adapt to network variability and improve congestion control performance.

The main advantage of ML-based approaches is their ability to learn from past behavior and predict future network states, which helps reduce oscillations and enhance steady-state performance. However, ML-based congestion control approaches also face some challenges, including difficulties in managing multiple optimization objectives and generalizing across diverse applications and network conditions [39]. Moreover, they often require substantial storage and processing power [14].

To address these limitations, recent trends suggest combining rule-based congestion control with DRL-based methods to leverage the strengths of both paradigms [14]. Hybrid approaches, as explored in [13,15,37], demonstrate that integrating ML with real-time adaptive rule-based methods can achieve more robust and adaptive congestion control, improving performance in dynamic and unpredictable network environments.

2.2. Queueing Theory and Optimality

The application of queueing theory to TCP congestion control has been significantly influenced by Kleinrock's optimality principle [17]. According to Kleinrock, a queueing system achieves optimal performance when its average occupancy equals one item. In TCP/IP networks, this corresponds to maintaining one BDP of data-in-flight. The BDP, calculated as bandwidth × two-way PropDelay, represents the amount of data that can traverse the network without requiring buffering.

To align with this principle, an optimal congestion control algorithm must adjust the sending rate so that the average data-in-flight remain close to one BDP, ensuring the queue remains nearly empty. TCP BBR, for example, was heavily influenced by Kleinrock's principle and limits data-in-flight to 2 × BDP to maintain at least one BDP in flight [24,40]. This approach maximizes throughput and minimizes delay by probing the maximum available bandwidth and probing for minRTT.

Beyond Kleinrock's work, Stidham's studies on optimal queueing system design [41,42] provide additional theoretical grounding. Stidham's framework emphasizes balancing

network utilization with congestion costs, identifying a point of diminishing returns where increasing throughput is no longer justified by the additional congestion incurred.

Together, the insights from Kleinrock and Stidham offer valuable guidance for designing optimal congestion control mechanisms. In this article, these principles are used to define the optimal operating range for data-in-flight as between 1 × BDP and 2 × BDP. Similarly, the optimal RTT range is set between 1 × baseRTT (baseRTT is the lowest measured RTT) and 2 × baseRTT, ensuring efficient performance while avoiding excessive queuing delays.

3. Background Concepts

3.1. End-to-End TCP Congestion Control

A network consists of multiple nodes and paths that facilitate data transfer between senders and receivers. The network's behavior is defined by several key characteristics: BtlBW, PropDelay, and buffer size, as illustrated in Figure 1 (see also [14]).

- BtlBW: The smallest bandwidth along a network path, which constrains the maximum achievable throughput.
- PropDelay: The one-way time taken for a packet to travel from the sender to the receiver when there is no congestion. It is determined by the physical distance, the transmission medium's speed and the processing delay. This delay reflects the minimum achievable time, unaffected by queuing.
- Buffer size: The capacity of network devices to temporarily store packets waiting for transmission, helping absorb transient congestion.

The BDP defines the amount of in-flight data that can traverse the network without requiring buffering. An end-to-end TCP CCA operates on the sender side , and it adjusts the sending rate to prevent the network congestion based on inferred conditions, as shown in Figure 1. In TCP, each end-point in a pair operates as a sender and a receiver at the same time. However, each independently manages its outgoing flow, applying congestion control without visibility into the other's behavior.

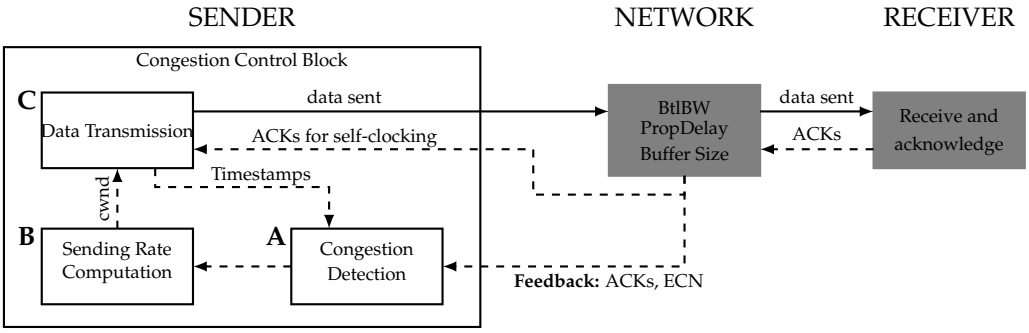

Figure 1. A block diagram for end-to-end TCP congestion control.

The congestion control block at the sender includes three main components:

- **A. Congestion Detection:** Identifies congestion through duplicate Acknowledgements (ACKs), timeouts, ECN, or delay measurements.
- **B. Sending Rate Computation:** Adjusts the sending rate based on feedback from congestion detection.
- **C. Data Transmission:** Sends data according to the computed rate, regulated by the congestion window size (cwnd), with the sending rate approximated by cwnd/RTT.

The receiver sends ACKs to provide feedback about received packets. These ACKs are also used for self-clocking, triggering the next data transmission and ensuring reliable,

ordered delivery. In TCP congestion control, various parameters, such as RTT, bandwidth estimates, and data-in-flight, are derived from the timing of received ACKs.

3.2. minRTT and baseRTT

TCP CCAs frequently rely on measurements of minRTT and baseRTT to infer network conditions. Although these terms are sometimes used interchangeably, they represent distinct concepts:

- minRTT: The lowest RTT observed by the sender within a specified time window.
- baseRTT: The absolute minimum RTT recorded throughout the lifetime of the connection, representing the network's two-way PropDelay under ideal conditions.

Under uncongested network conditions, minRTT often aligns with baseRTT. Algorithms such as TCP Vegas detect congestion by comparing the current RTT with baseRTT [43]. Similarly, TCP BBR periodically probes for minRTT (referred to as rtProp) to detect changes in PropDelay over time and adjust its sending rate accordingly.

In this article, minRTT is used in a manner similar to TCP BBR, serving as a critical metric for congestion avoidance and performance optimization.

3.3. Queueing Theory and TCP Congestion Control

Following Stidham's work [41,42], TCP/IP networks can be modeled as a queueing system [44]. As shown in Figure 2, the system comprises the following:

- Waiting Area: Represents the network buffer with capacity B.
- Service Node: Processes data at the network bandwidth capacity μ, equivalent to the BtlBW.
- Response Time: The minimum response time is approximated by baseRTT or minRTT.

This model helps explain network behavior during congestion.

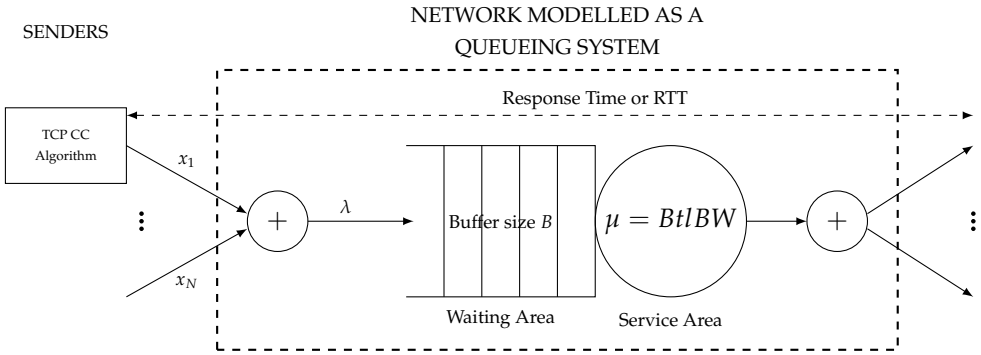

Figure 2. A queueing system model for an end-to-end TCP/IP network.

In this queueing model, data packets arrive at an arrival rate λ and are served at the network bandwidth capacity μ. The goal of congestion control is to regulate the sending rates of multiple flows $x_1, x_2, \ldots, x_N$ to match the network capacity and prevent buffer overflow or excessive queuing delays [41].

Key performance metrics used in queueing theory and congestion control [17,41,45–47] include the following:

- **Occupancy** L: The number of items in the system, including those waiting and being served. It corresponds to data-in-flight in TCP.
- **Arrival rate** λ: The total rate at which items arrive in the queue, equivalent to the sum of the sending rates.
- **Throughput**: The rate at which data are delivered over a network link.

- **Goodput**: The rate of useful data delivery, excluding retransmissions [19]. It is a reliable indicator of effective utilization. It reflects the network's productive performance.
- **Response time** R: In queueing theory, it refers to the duration from the moment an item enters the system to the time a response is received upon the completion of its processing. In TCP it is RTT.
- **Utilization** $\rho = \lambda/\mu$: The ratio of arrival rate λ to the network bandwidth capacity μ in a queueing system.
- **Fairness**: Equitable distribution of resources.

In both literature and practice, these metrics are commonly expressed as average values (e.g., average occupancy, average arrival rate). For brevity, this article omits the term 'average' unless its inclusion is necessary for clarity or emphasis.

3.4. Achieving Fairness

Jain's Fairness Index is a widely recognized metric for quantifying fairness in resource allocation [47,48]. It is defined as follows:

$$J = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \sum_{i=1}^{N} x_i}, \tag{1}$$

where x_i is the throughput of flow i, and N is the total number of flows. A value of $J = 1$ indicates perfect fairness, while $J = 1/N$ signifies the poorest fairness. In practice, a fairness index of 0.8 is acceptable [49].

Achieving fairness in network congestion control is critical to ensuring equitable bandwidth distribution among competing data flows. Fairness prevents any single flow from monopolizing network resources, thereby improving overall network efficiency and user satisfaction. This article focuses specifically on intra-fairness, which refers to fairness among flows using the same congestion control algorithm.

AIMD [50] is a foundational mechanism in TCP congestion control, extensively analyzed for its fairness properties [51,52]. AIMD achieves fairness by gradually converging to an equitable allocation of resources during its multiplicative decrease phase, which reduces the sending rate in response to congestion signals. This reduction allows other flows to increase their share of bandwidth, fostering balanced resource distribution. Studies in [51,52] identify two primary factors influencing fairness in AIMD-based algorithms:

- Frequency of the Decrease Phase: Higher frequencies accelerate convergence to fairness as congestion signals are processed more often, though they may introduce instability.
- Amplitude of Oscillations: Smaller oscillations lead to higher bandwidth utilization but slower convergence to fairness.

These trade-offs highlight AIMD's ability to balance fairness with resource efficiency, though it requires careful tuning of its increase and decrease parameters.

Research by [53] has shown that algorithms incorporating nonlinear increase strategies, such as Multiplicative Increase, Multiplicative Decrease (MIMD), can converge to fairness faster than AIMD under certain conditions. For instance, TCP CUBIC [10], which employs a cubic window growth function, achieves high throughput while maintaining good convergence to fairness due to its aggressive probing mechanism.

TCP BBR (Version 1) adopts a fundamentally different approach by adjusting its sending rate based on measurements of available bandwidth and the minRTT. Unlike traditional loss-based approaches, BBR does not reduce its sending rate in response to packet loss. Instead, it periodically reduces the sending rate to probe for minRTT. This periodic reduction creates opportunities for competing flows to claim available bandwidth, enabling fairness.

Simulation studies [54,55] indicate that multiple TCP BBR flows with similar minRTTs can achieve intra-fairness. However, sustained unfairness may arise due to mismeasurements of available bandwidth or significant RTT variations. The focus on TCP BBR's intra-fairness is particularly relevant to this article, as the proposed mechanism also leverages a minRTT probing strategy. Understanding TCP BBR's strengths and limitations in achieving fairness provides valuable insights for improving the design and evaluation of the proposed approach.

4. Proposed Congestion Control Mechanism

4.1. Corollary from Little's Law

Little's Law [26] states the relationship between the arrival rate λ, response time $R(\lambda)$, and occupancy $L(\lambda)$ as

$$L(\lambda) = \lambda R(\lambda). \tag{2}$$

Differentiation of $L(\lambda)$ with respect to λ yields

$$\frac{dL(\lambda)}{d\lambda} = \lambda \frac{dR(\lambda)}{d\lambda} + R(\lambda) \tag{3}$$

Thus, the rate of change in $R(\lambda)$ with respect to λ is

$$R'(\lambda) = \frac{L'(\lambda) - R(\lambda)}{\lambda}. \tag{4}$$

When $R'(\lambda) \to 0$, the system operates near an optimal point, implying that

$$L'(\lambda) - R(\lambda) \approx 0. \tag{5}$$

A key corollary from (5) is that optimal performance is achieved when $L'(\lambda)$ approaches a target response time R_{target}. If the lowest observed RTT (either minRTT or baseRTT) is denoted as R_{min}, then

$$R_{\text{target}} = \alpha R_{\text{min}}, \quad \alpha > 1, \tag{6}$$

where α is a multiplicative factor determining leeway from minRTT. Practical experiments suggest that values around $\alpha = 1.2$ offer stability, whereas lower values lead to instability, and higher values increase oscillations. More work is required to determine an optimal value.

4.2. Numerical Computational Framework

Equation (5) represents an ordinary differential equation that behaves as a harmonic oscillator, which can be effectively solved using the numerical direct shooting method [56,57]. This approach involves discretizing the differential equation and iteratively solving the resulting boundary value problem by adjusting initial conditions. The solution process is inductive and iterative, refining the initial guess step-by-step to ensure the boundary conditions are met accurately.

The discrete form of Equation (5) can be expressed as

$$\frac{L_{k+1} - L_k}{\lambda_{k+1} - \lambda_k} - R_{k+1} = 0. \tag{7}$$

where k and $k+1$ denote consecutive time steps. In the context of TCP congestion control, the sending rate is estimated as W/R, where W is the cwnd and R is RTT. Substituting this approximation into Equation (7), we obtain the following:

$$\frac{L_{k+1} - L_k}{W_{k+1}/R_{k+1} - W_k/R_k} - R_{k+1} = 0. \tag{8}$$

A key insight of this formulation is that it eliminates the need to calculate the BDP explicitly, allowing L and W to remain in natural units (e.g., packets), avoiding additional bandwidth estimation overhead.

In TCP congestion control, the cwnd W is the primary control variable. The values of L (data-in-flight) and R (RTT) respond dynamically to changes in W, with their measurements directly obtained from the system once a particular W is applied. Notably, R is a responsive variable that reflects the influence of L; as L increases, queueing delays can accumulate, resulting in higher RTTs.

To optimize the system, two algebraic equations are employed: one predicts the future value of L, while the other computes the optimal W based on that prediction. These equations are solved iteratively using a boundary value approach. The system reaches an optimal operating point when

$$R' \approx 0, \quad R \approx R_{\text{target}}, \text{ and } \quad L \approx \mu R_{\text{target}}$$

Predicting the Future Value of L:

The next task is to predict the value of L at the upcoming time step. Assume that if W remains unchanged during the iteration cycle, L will evolve from L to L_{predict}. Due to the harmonic dynamics of the system, if R is adjusted to R_{target}, it will oscillate back to its original value by the time L reaches L_{predict}, provided W remains constant. Therefore, the corresponding collocation points are as follows:

$$(L, R_{\text{target}}, W) \text{ at instance } k, \text{ and}$$
$$(L_{\text{predict}}, R, W) \text{ at instance } k+1$$

Substituting these values into Equation (8), we derive the following:

$$\frac{L_{\text{predict}} - L}{W/R - W/R_{\text{target}}} - R = 0, \tag{9}$$

which simplifies to the following:

$$L_{\text{predict}} = L + W - (R/R_{\text{target}})W. \tag{10}$$

This prediction model ensures convergence to the target RTT, R_{target}, based on the following observations:

- If $R < R_{\text{target}}$: The predicted value L_{predict} increases, encouraging a higher cwnd.
- If $R > R_{\text{target}}$: L_{predict} decreases, prompting a reduction in cwnd.
- If $R = R_{\text{target}}$: $L_{\text{predict}} = L$, indicating a stable state.

This mechanism actively regulates the evolution of L toward a steady state, preventing oscillations.

Computing the Next Optimal Value of W:

Once L_{predict} is known, the next optimal cwnd, denoted W_{new}, can be computed. Suppose that during this iteration, the cwnd changes from W to W_{new} while L evolves to L_{predict} in response. Assuming the RTT remains at $R_{\min}$ (the minimal observed RTT) until queueing begins, the collocation points are given by the following:

(L, R_{min}, W) at instance k,
$(L_{\text{predict}}, R_{\text{min}}, W_{\text{new}})$ at an instance between k and $k+1$, and
$(L_{\text{predict}}, R, W_{\text{new}})$ at instance $k+1$

Substituting into Equation (8) and assuming that the change from R_{min} to R is just before the end of the cycle, we obtain the following:

$$\frac{L_{\text{predict}} - L}{W_{\text{new}}/R_{\text{min}} - W/R_{\text{min}}} - R = 0. \tag{11}$$

Solving for W_{new}, we obtain the following:

$$W_{\text{new}} = W + (R_{\text{min}}/R)(L_{\text{predict}} - L). \tag{12}$$

This adjustment mechanism ensures that

- If $R < R_{\text{target}}$: $L_{\text{predict}} > L$, resulting in an increase in W.
- If $R > R_{\text{target}}$: $L_{\text{predict}} < L$, causing W to decreases.
- The further R deviates from R_{min}: The rate of increase in W becomes more gradual.
- If $R = R_{\text{target}}$: $L_{\text{predict}} = L$ and W remains unchanged.

Together, Equations (11) and (12) act as inertia and damping mechanisms, balancing the interaction between data-in-flight, RTT, and cwnd. This iterative process continues until the system converges to the optimal equilibrium, where $R = R_{\text{target}}$.

This numerical framework harmonizes opposing forces within the congestion control mechanism through collocation-based prediction and control. By leveraging real-time feedback from network conditions and adjusting the cwnd iteratively, the system ensures stable operation at optimal conditions without requiring explicit BDP calculations. This method offers a scalable, responsive approach to managing congestion, providing smooth network performance across varying conditions.

4.3. Implementation in ns-3

The proposed congestion control mechanism is implemented and visualized in the block diagram in Figure 3. Each iteration involves several coordinated steps to monitor and adjust network parameters in real time.

Block A: This block gathers the current values of key network metrics:

- Congestion Window W
- Data-in-Flight L
- RTT R
- minRTT or baseRTT (R_{min})

Block B: Computes essential targets and predictions using the numerical framework:

- R_{target}: Target RTT computed from R_{min}.
- L_{predict}: Future data-in-flight based on current conditions.
- W_{new}: Updated congestion window.

Block C: Uses W_{new} to regulate the data sending rate, ensuring that network resources are used efficiently while avoiding congestion.

The mechanism was implemented using ns-3.41 to evaluate both basic and practical network conditions (see Appendix B). Two variants, TCP QtCol and TCP QtColFair, are provided to explore the effectiveness of the proposed scheme under different scenarios. These implementations are available on GitHub along with visualization and computational tools using Python [58].

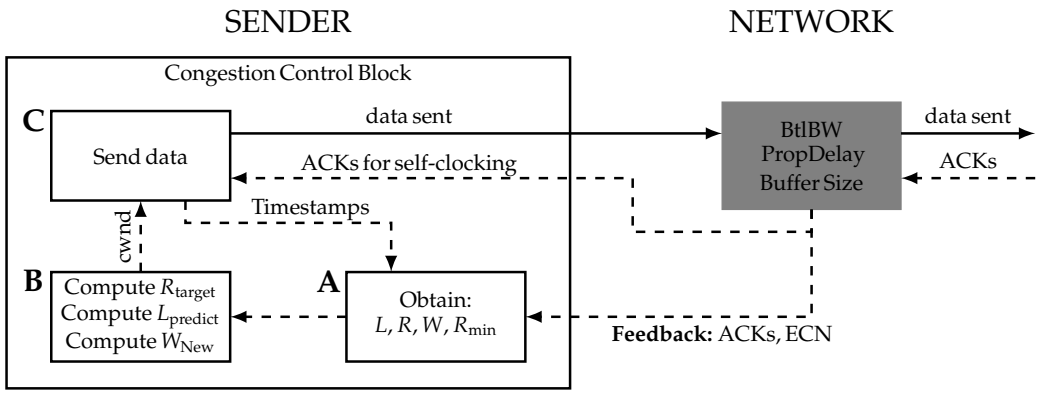

Figure 3. A block diagram for the proposed end-to-end TCP congestion control mechanism.

4.3.1. Basic Implementation

The TcpQtCol class, shown in Figure 4, extends the TcpNewReno class in ns-3. The pseudocode in Algorithm 1. This variant assumes constant network conditions without considering fairness or varying PropDelay. It uses baseRTT instead of minRTT to define R_{min}.

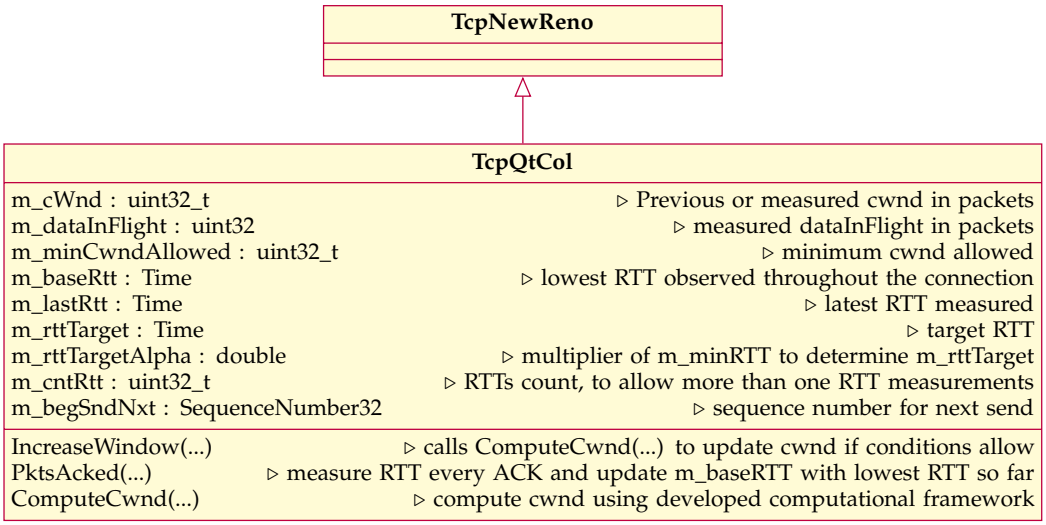

Figure 4. TCP QtCol UML class diagram for basic implementation.

Algorithm 1 Compute cwnd pseudocode for the basic implementation

Require: m_cWnd, m_dataInFlight, m_lastRtt, m_minRtt, m_rttTargetAlpha > 1, m_minCwndAllowed
  $W \leftarrow$ m_cWnd, $L \leftarrow$ m_dataInFlight
  $R \leftarrow$ m_lastRtt, $R_{min} \leftarrow$ m_minRtt
  $\alpha \leftarrow$ m_rttTargetAlpha
  $R_{target} \leftarrow \alpha R_{min}$ ▷ target RTT
  $L_{pred} \leftarrow \max(L + W - W \times R / R_{target}, \text{m\_minCwndAllowed})$
  if $L_{pred} = L$ **then**
    $L_{pred} \leftarrow L + 1$ ▷ allow consistent variability
  end if
  $W_{new} \leftarrow \max(W + \lceil R_{min} \times (L_{pred} - L)/R \rceil, 4)$ ▷ err on the side of increasing to avoid saturation
  if $W_{new} = W$ **then**
    $W_{new} \leftarrow W + 1$ ▷ allow consistent variability
  end if

4.3.2. Practical Implementation

TCP QtColFair adapts the congestion control mechanism to real-world conditions, such as dynamic PropDelay and fairness across multiple flows. It uses minRTT with periodic probing, similar to TCP BBR, to track network changes. If RTT increases unexpectedly, the algorithm adjusts cwnd to reflect the new network state. The implementation is described using TcpQtColFair class in Figure 5 and pseudocode in Algorithm 2.

Handling RTT Probing and Fairness:

- Probing Phase: Every 10 s, cwnd is reduced to four packets for 200 ms to probe for the current minRTT.
- Fairness Enforcement: RTT probing contributes towards fairness. In addition, setting L_{predict} proportionately lower after probing ensures high-throughput flows back off temporarily to allow other flows to increase their rates
- Dynamic Delay Handling: If the PropDelay changes unexpectedly, the algorithm reduces data-in-flight and waits for RTT stability before increasing cwnd again.

These techniques ensure fairness and responsiveness even in dynamic network environments.

Algorithm 2 Compute cwnd pseudocode for the practical implementation

Require: m_cWnd, m_dataInFlight, m_priorCwnd, m_minCwndAllowed
Require: m_lastRtt, m_minRtt, m_targetRtt, m_rttTargetAlpha > 1, m_probeRtt, m_probeRttRecover
  $\alpha \leftarrow$ m_rttTargetAlpha
  if m_probeRtt $= 0$ & m_dataInFlight $< 0.5 \times$ m_priorCwnd & m_lastRtt $> 2 \times$ m_targetRtt **then**
    m_minRttChangeCnt $\leftarrow$ m_minRttChangeCnt $+ 1$
    if m_minRttChangeCnt > 2 **then**
      m_probeRtt $\leftarrow 1$
      m_minRtt $\leftarrow$ m_lastRtt ▷ quickly dump old m_minRtt if change confirmed
      m_targetRtt $= \alpha \times$ m_minRtt
    end if
  else
    m_minRttChangeCnt $\leftarrow 0$
  end if
  $W \leftarrow$ m_cWnd, $L \leftarrow$ m_dataInFlight, $R \leftarrow$ m_lastRtt, $R_{\text{min}} \leftarrow$ m_minRtt, $R_{\text{target}} \leftarrow \alpha R_{\text{min}}$
  if m_probeRtt **then**
    $W_{\text{new}} \leftarrow$ m_minCwndAllowed
  else
    if m_probeRttRecover **then**
      $L_{\text{pred}} \leftarrow 0.9 \times$ m_priorCwnd ▷ quickly return to previous state after probing
      m_probeRttRecover $\leftarrow 0$
    else
      $L_{\text{pred}} \leftarrow L + W - W \times R / R_{\text{target}}$
    end if
    $L_{\text{pred}} \leftarrow \max(L_{\text{pred}}, \text{m\_minCwndAllowed})$
    if $L_{\text{pred}} = L$ **then**
      $L_{\text{pred}} \leftarrow L + 1$ ▷ allow consistent variability
    end if
    $W_{\text{new}} \leftarrow W + \lceil R_{\text{min}} \times (L_{\text{pred}} - L) / R \rceil$ ▷ err on the side of increasing to avoid saturation
    $W_{\text{new}} \leftarrow \max(W_{\text{new}}, \text{m\_minCwndAllowed})$
    if $W_{\text{new}} = W$ **then**
      $W_{\text{new}} \leftarrow W + 1$ ▷ allow consistent variability
    end if
  end if

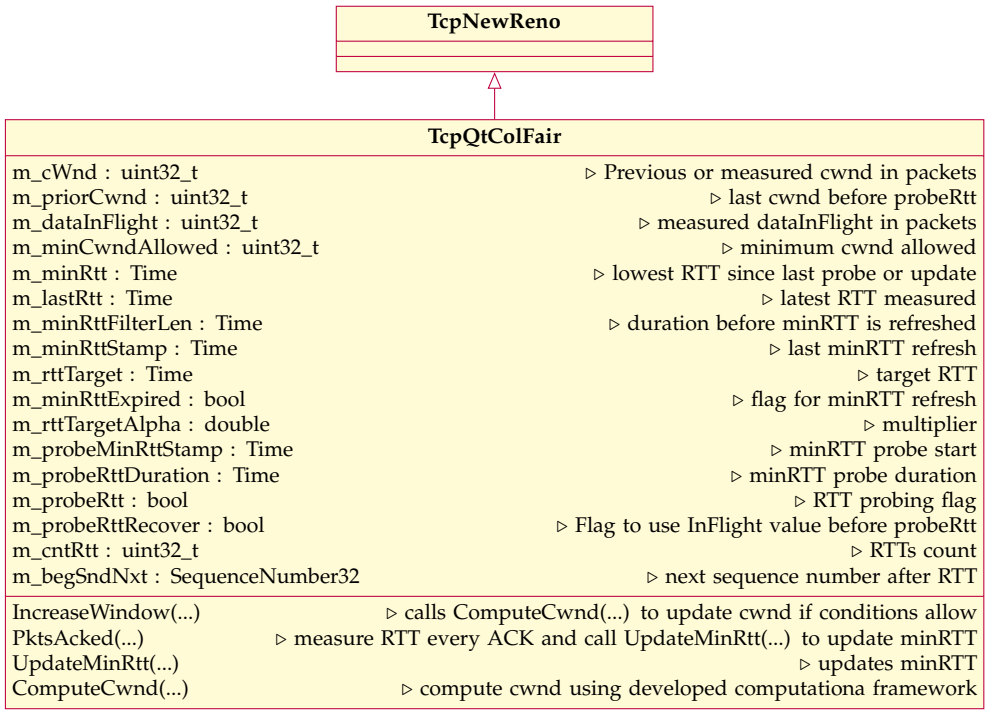

Figure 5. TcpQtColFair UML class diagram for practical implementation.

5. Simulations and Results

5.1. Simulations Model

Simulations were conducted using ns-3 and the topology shown in Figure 6 with associated configurations. The topology was chosen so that all end systems share the same bottleneck link.

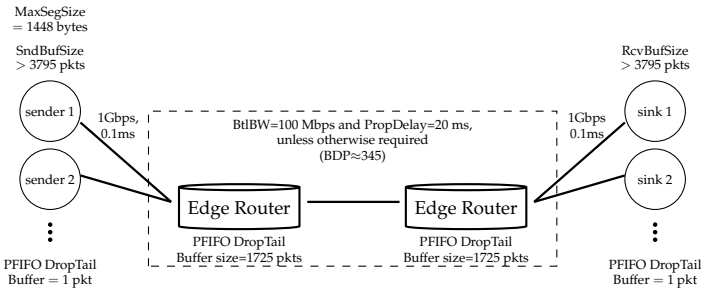

Figure 6. Network topology used for simulations.

The TCP maximum segment size (MaxSegSize) was set to 1448 bytes, based on Ethernet's 1500-byte Maximum Transmission Unit (MTU), subtracting 52 bytes for TCP and IP headers [59,60]. The BtlBW and PropDelay were configured at 100 Mbps and 20 ms, respectively, resulting in a BDP of approximately 345 packets. The network buffer size was set to 1725 packets (5 × BDP), ensuring sufficient capacity to test the system's ability to avoid queueing delays and packet loss.

Access links were assigned 1 Gbps bandwidth and 0.1 ms PropDelay to ensure the bottleneck link remained the primary constraint. In certain simulations, the buffer size was varied between 1 × BDP and 10 × BDP, allowing for the exploration of different congestion scenarios. For a buffer size of 10 × BDP, the maximum possible cwnd was 11 × BDP = 3795 packets. To prevent buffer constraints from limiting throughput, the SndBufSize and RcvBufSize were set well above 3795 packets.

Simulations tested TCP QtCol and TCP QtColFair with α values of 1.2 and 1.5, representing small and large values, respectively. Scenarios included

- Single-flow static: Ideal network conditions.
- Single-flow non-static: Varying bandwidth or PropDelay.
- Multiple-flow scenarios: Evaluating buffer size impact and flow fairness.

5.2. Single-Flow Static Scenario—Delay Avoidance, Kleinrock's Optimality, and Oscillations

This section evaluates the proposed algorithm's ability to optimize data-in-flight and RTT while eliminating oscillations, compared with TCP CUBIC and TCP BBR.

As shown in Figures 7 and 8, TCP CUBIC struggles to control data-in-flight and RTT effectively due to its cubic growth function, which causes oscillations between 1449 and 2070 packets, and RTT fluctuations between 0.17 and 0.25 s. TCP CUBIC's oscillations result from cwnd increasing until the link's carrying capacity (BDP + buffer size) is reached, followed by a 30% drop.

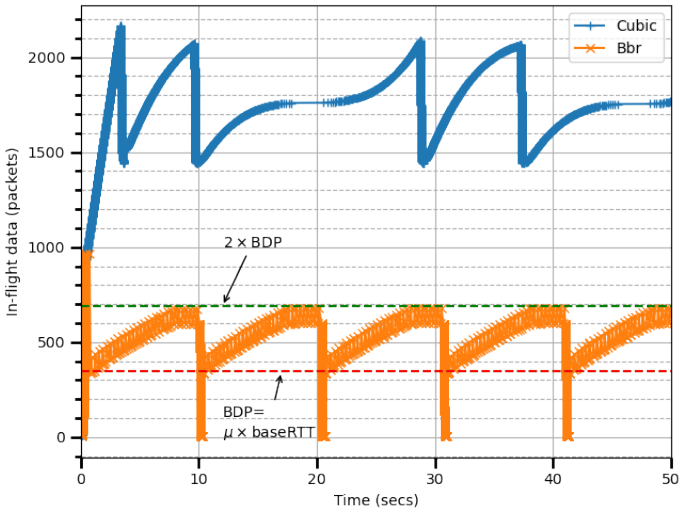

Figure 7. Data-in-flight for TCP CUBIC and TCP BBR in a single-flow scenario. Values within the range of BDP and 2 × BDP are considered acceptable.

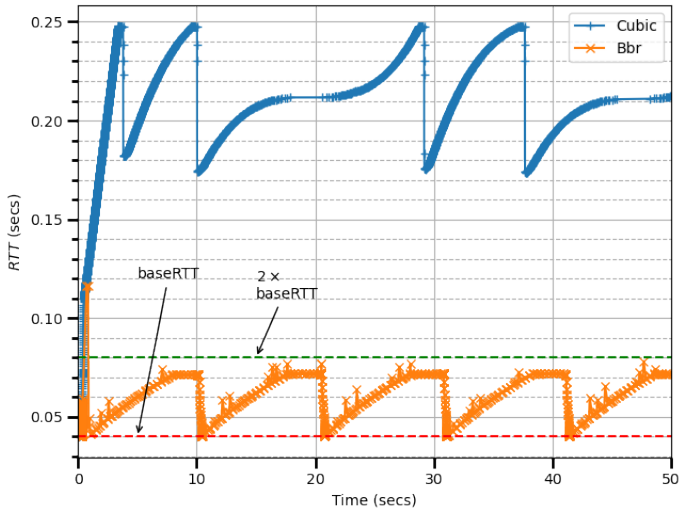

Figure 8. RTT or response time for TCP CUBIC and TCP BBR in a single-flow scenario. Values within the range of baseRTT and 2 × baseRTT are considered acceptable.

In contrast, TCP BBR keeps data-in-flight near the BDP (minRTT × BtlBW), independent of buffer size. However, bandwidth and RTT probing mechanisms cause oscillations between 1 × BDP and 2 × BDP (345 to 690 packets) and RTT between baseRTT and 2 × baseRTT (0.04 to 0.08 s).

TCP QtCol performs similarly to TCP BBR in controlling data-in-flight and RTT, as shown in Figures 9 and 10, but with dampened oscillations. For α values of 1.2 and 1.5, data-in-flight converge to 400 and 500 packets, respectively, (below 2 $\times$ BDP) and RTT stabilizes at 48 ms and 60 ms. TCP QtCol achieves equilibrium faster than TCP BBR and maintains stable performance without oscillations under steady conditions.

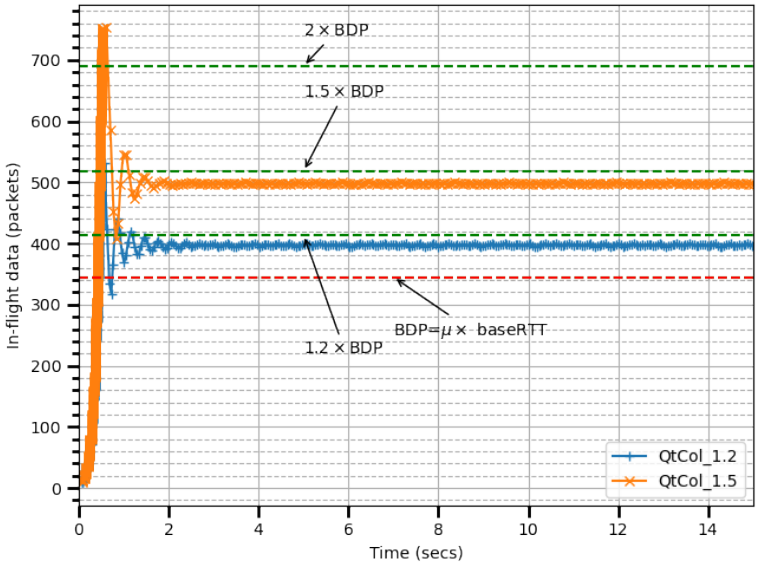

Figure 9. Data-in-flight for TCP QtCol in a single-flow scenario. Values within the range of BDP and 2 $\times$ BDP are considered acceptable. The line labeled BDP = μ $\times$ baseRTT represents the convergence value of data-in-flight when the RTT equals the baseRTT. The lines labeled 1.2 $\times$ BDP and 1.5 $\times$ BDP represent the expected convergence values of data-in-flight when the target RTT is 1.2 $\times$ baseRTT and 1.5 $\times$ baseRTT, respectively.

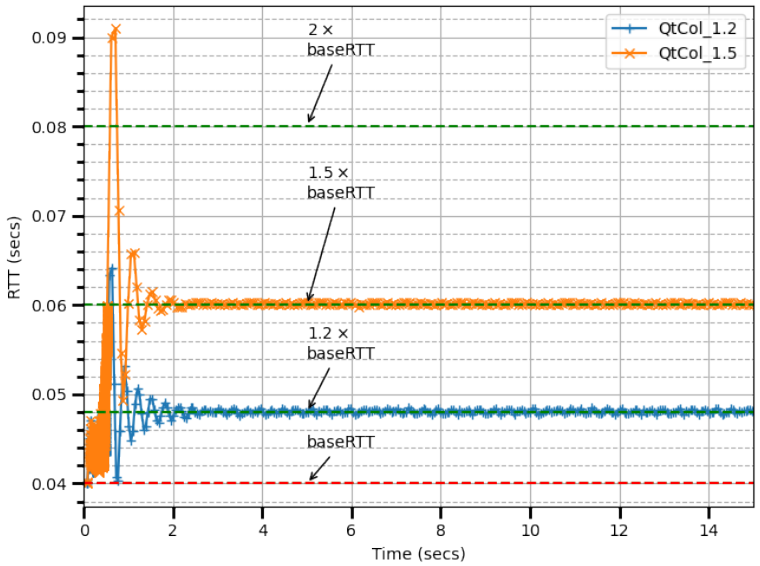

Figure 10. RTT or response time for TCP QtCol in a single-flow scenario. Values within the range of baseRTT and 2 $\times$ baseRTT are considered acceptable. The lines labeled 1.2 $\times$ baseRTT and 1.5 $\times$ baseRTT represent the expected convergence values of RTT when the target RTT is 1.2 $\times$ baseRTT and 1.5 $\times$ baseRTT, respectively.

TCP QtColFair extends TCP QtCol by incorporating additional mechanisms to handle real-world network challenges. As shown in Figures 11 and 12, with $\alpha = 1.2$, data-in-flight stay near 1.2 $\times$ BDP, and RTT remains close to 1.2 $\times$ baseRTT, immediately after

disturbances. TCP QtColFair quickly dampens oscillations that may arise from external factors, outperforming both TCP CUBIC and TCP BBR.

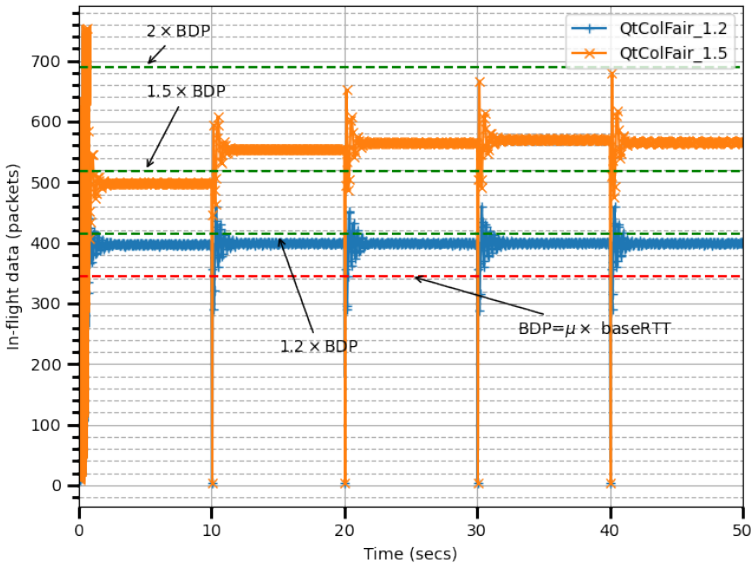

Figure 11. Data-in-flight for TCP QtCol in a single-flow scenario.Values within the range of BDP and 2 × BDP are considered acceptable. The line labeled BDP = μ × baseRTT represents the convergence value of data-in-flight when the RTT equals the baseRTT. The lines labeled 1.2 × BDP and 1.5 × BDP represent the expected convergence values of data-in-flight when the target RTT is 1.2 × baseRTT and 1.5 × baseRTT, respectively.

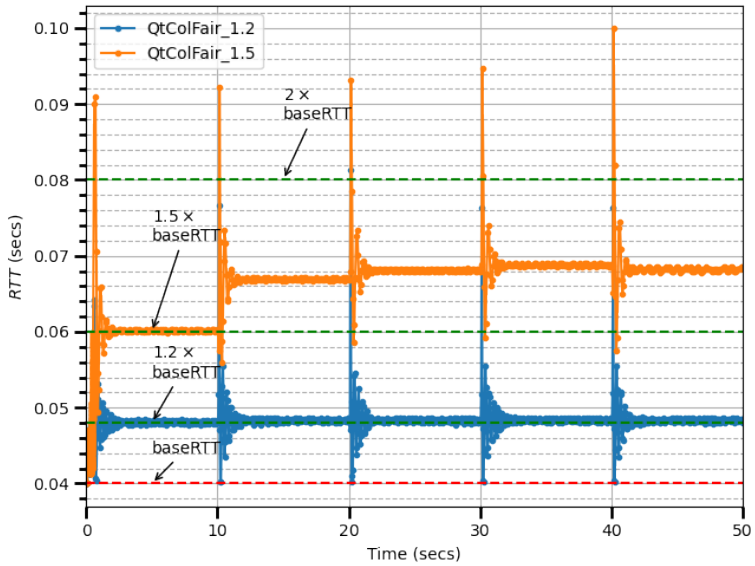

Figure 12. RTT or response time for TCP QtCol in a single-flow scenario. Values within the range of baseRTT and 2 × baseRTT are considered acceptable. The lines labeled 1.2 × baseRTT and 1.5 × baseRTT represent the expected convergence values of RTT when the target RTT is 1.2 × baseRTT and 1.5 × baseRTT, respectively.

The box-plot analysis (a description of a box-plot is given in Appendix A) in Figures 13 and 14 further illustrates that TCP CUBIC's data-in-flight and RTT are consistently outside the optimal range. TCP BBR achieves Kleinrock's optimality on average but with significant oscillations in data-in-flight. In contrast, TCP QtCol and TCP QtColFair exhibit narrow interquartile ranges (IQR), indicating precise control over data-in-flight and RTT. With $\alpha = 1.2$, TCP QtColFair achieves RTT values closest to baseRTT.

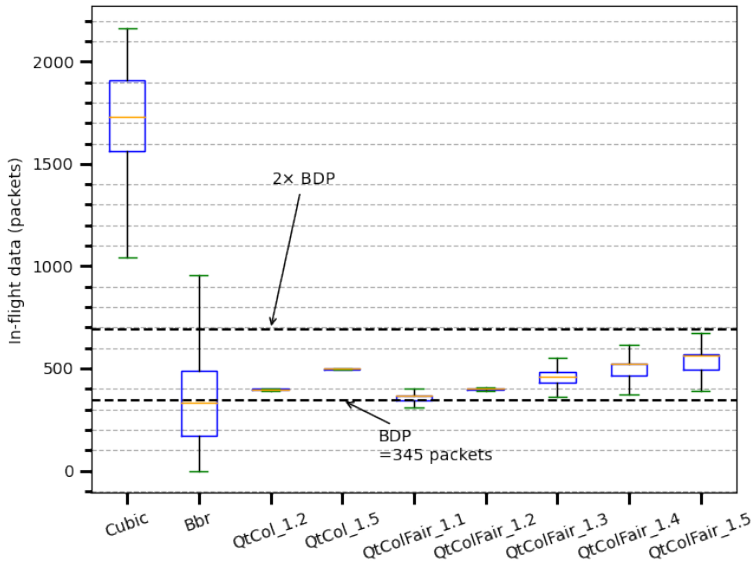

Figure 13. Data-in-flight box-plot for TCP QtCol and TCP QtColFair compared with TCP CUBIC and TCP BBR in a single-flow scenario. Values within the range of BDP and 2 × BDP are considered acceptable.

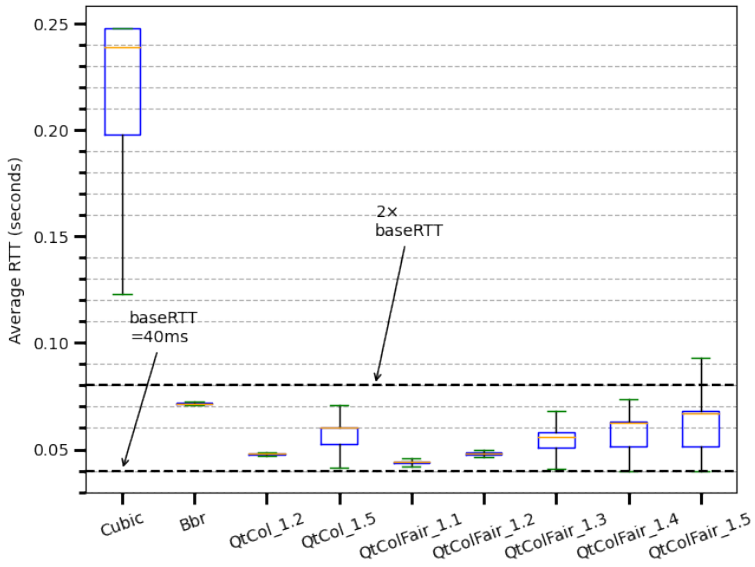

Figure 14. RTT box-plot for TCP QtCol and TCP QtColFair compared with TCP CUBIC and TCP BBR in a single-flow scenario. Values within the range of baseRTT and 2 × baseRTT are considered acceptable.

In summary, TCP QtCol and TCP QtColFair provide better RTT control, oscillation elimination, and optimization of data-in-flight compared to TCP CUBIC and TCP BBR, making them more effective at avoiding delays and ensuring smooth network performance.

5.3. Single-Flow Non-Static—Varying BtlBW

This section evaluates the ability to maintain low RTT under changing BtlBW conditions. The BtlBW was set to 100 Mbps for 25 s, reduced to 25 Mbps, and later increased to 500 Mbps. The PropDelay remained at 20 ms throughout.

When bandwidth increases, queues may accumulate, requiring the source to reduce its sending rate to avoid congestion. Figure 15 shows that TCP CUBIC struggles, with RTT varying from 40 to 480 ms, indicating poor control. TCP BBR performs better, keeping RTT between 45 and 58 ms. However, TCP QtCol and TCP QtColFair, with $\alpha = 1.2$, demonstrate

superior stability, maintaining RTT around 50 ms under all conditions, outperforming TCP BBR.

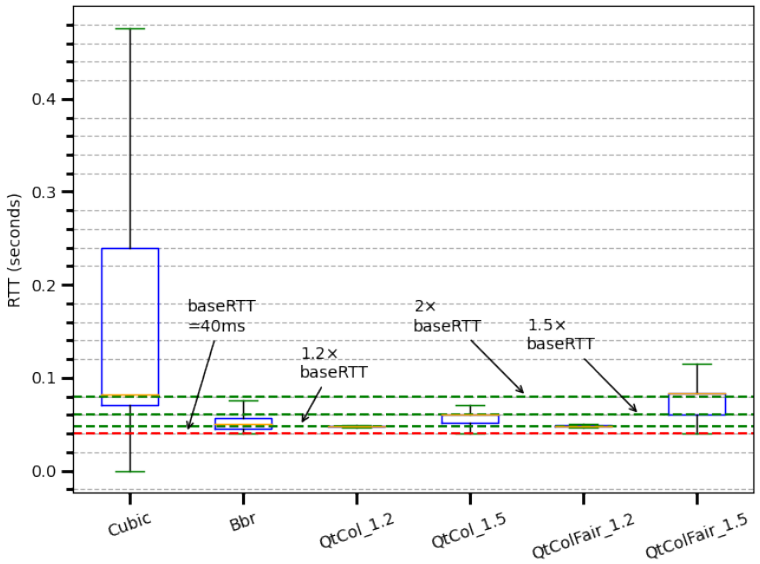

Figure 15. RTT for TCP CUBIC as BtlBW changes. Values within the range of baseRTT and 2 × baseRTT are considered acceptable. The lines labeled 1.2 × baseRTT and 1.5 × baseRTT represent the expected median RTT when the target RTT is 1.2 × baseRTT and 1.5 × baseRTT, respectively.

5.4. Single-Flow Non-Static—Varying PropDelay

This section demonstrates the importance of refreshing minRTT when PropDelay changes and evaluates TCP QtColFair's ability to adapt accordingly. The PropDelay was initially 20 ms, increased to 40 ms for 25 s, and then reverted to 20 ms.

TCP QtCol fails to detect changes in PropDelay, leading to a reduction in data-in-flight as it incorrectly tries to reduce RTT, as shown in Figure 16. This highlights TCP QtCol's limitations in practical networks. In contrast, TCP QtColFair effectively detects the change, maintaining higher data-in-flight during increased PropDelay, as shown in Figure 17. This capability makes TCP QtColFair better suited for real-world scenarios.

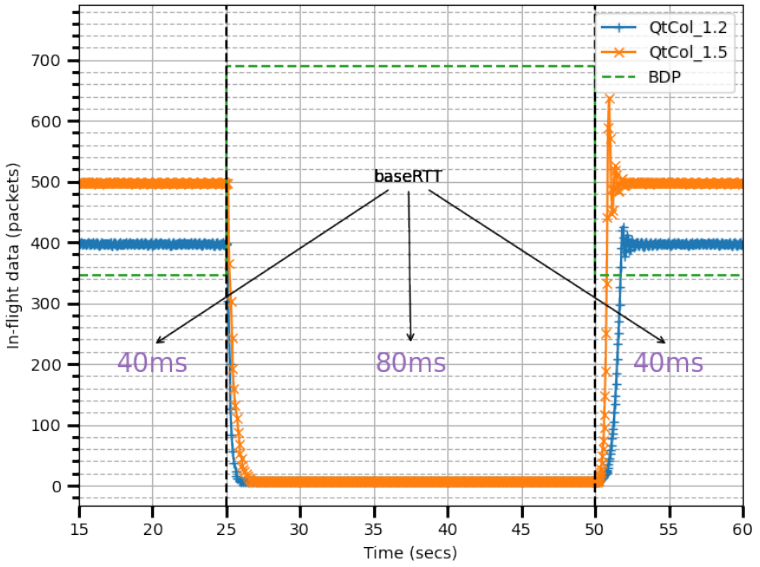

Figure 16. Data-in-flight for TCP QtCol when PropDelay changes.

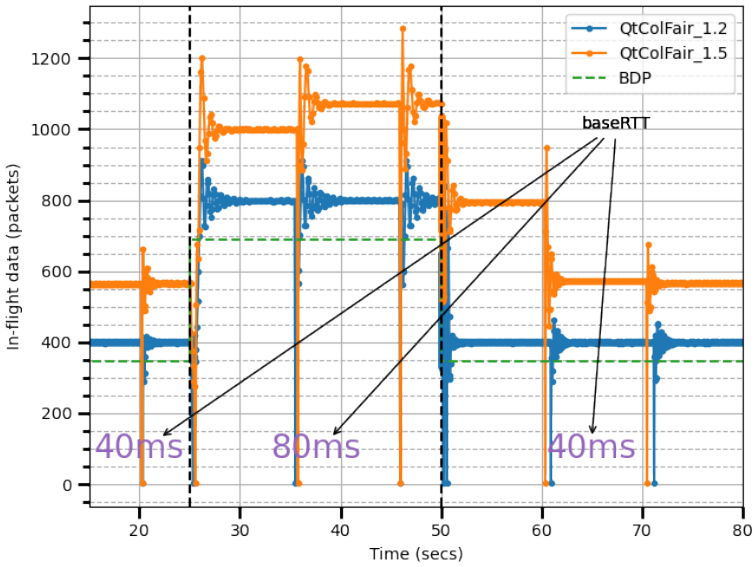

Figure 17. Data-in-flight for TCP QtColFair when PropDelay changes.

5.5. Multiple-Flow Scenario

This section evaluates the performance of TCP QtColFair in comparison to TCP CUBIC and TCP BBR within a practical multi-flow environment. The BtlBW was configured to 100 Mbps with a PropDelay of 20 ms, while the number of concurrent flows was incrementally increased from one to five. The analysis focused on the following metrics:

- **Optimization of Data-in-Flight:** Ensuring that the amount of in-flight data is tuned to prevent buffer overflow and mitigate congestion.
- **Queueing Delay Avoidance:** Minimizing RTT by reducing queueing delays.
- **Goodput:** Measuring effective network utilization, reflecting the proportion of transmitted data successfully delivered to the destination.
- **Intra-Fairness:** Ensuring an equitable distribution of bandwidth among flows using the same congestion control algorithm.

This experimental setup provides valuable insights into the algorithms' ability to manage congestion and fairness under realistic network conditions, highlighting their strengths and limitations in multi-flow scenarios.

5.5.1. Data-in-Flight Optimization and Delay Avoiding Capability

Figure 18 illustrates the algorithms' ability to optimize data-in-flight. TCP CUBIC's values fall outside the optimal range, with large IQR and oscillations, reflecting its loss-based nature. TCP BBR maintains 50% of its values within the optimal range but also exhibits high-amplitude oscillations. In contrast, TCP QtColFair shows better optimization, with most data-in-flight values concentrated around the median, especially for $\alpha \leq 1.2$, achieving a performance closest to Kleinrock's optimality.

Figure 19 shows the RTT performance. TCP CUBIC exhibits large oscillations, indicating poor delay control. TCP BBR performs better, but its RTT values remain a lot higher than minRTT, showing limited queue management. TCP QtColFair, particularly with smaller α values, maintains RTT closer to minRTT than TCP BBR, demonstrating superior queueing delay avoidance.

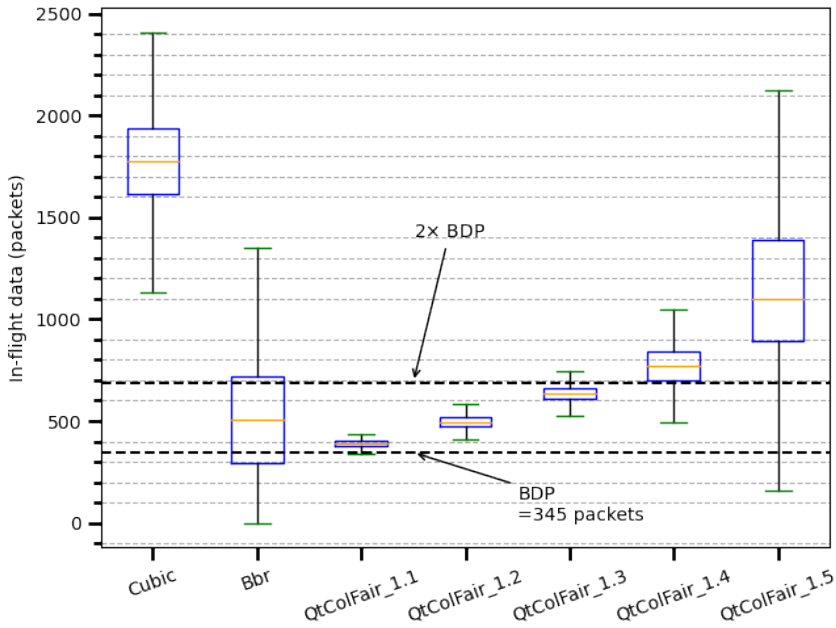

Figure 18. Data-in-flight for TCP QtColFair compared with TCP CUBIC and TCP BBR in multiple-flow scenario. Values within the range of BDP and 2 × BDP are considered acceptable.

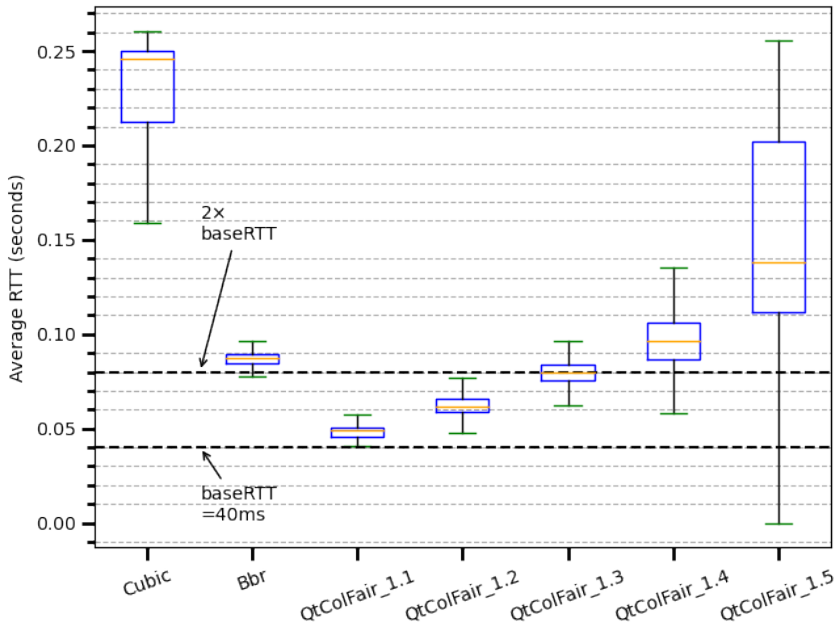

Figure 19. RTT for TCP QtColFair compared with TCP CUBIC and TCP BBR in multiple-flow scenarios. Values within the range of baseRTT and 2 × baseRTT are considered acceptable.

5.5.2. Effective Network Utilization

Figure 20 presents a comparison of goodput and network utilization across the algorithms in a multi-flow scenario. TCP QtColFair achieves nearly 96% utilization, outperforming both TCP BBR (just above 94%) and TCP CUBIC (around 93%). TCP CUBIC's utilization further declines as the network buffer size increases, indicating poor adaptability in high-buffer environments. TCP BBR performs more consistently but is still outmatched by TCP QtColFair, which offers superior network efficiency under these conditions.

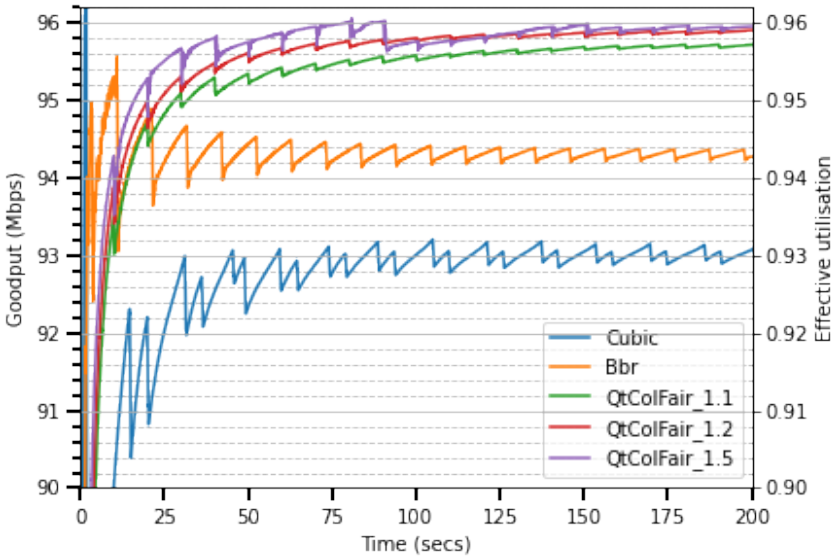

Figure 20. Effective network utilization and goodput for the algorithms in multiple-flow scenarios.

5.5.3. Fairness Between Flows of the Same Type

Figure 21 compares the algorithms using Jain's fairness ratio. All algorithms (TCP QtColFair, TCP CUBIC, and TCP BBR) achieve ratios above 0.9, indicating high intra-fairness and convergence toward an ideal value of 1 over time. In contrast, TCP QtCol performs poorly, with a fairness ratio of around 0.2 (equal to 1/N for N = 5 flows), demonstrating its inability to distribute bandwidth fairly across competing flows.

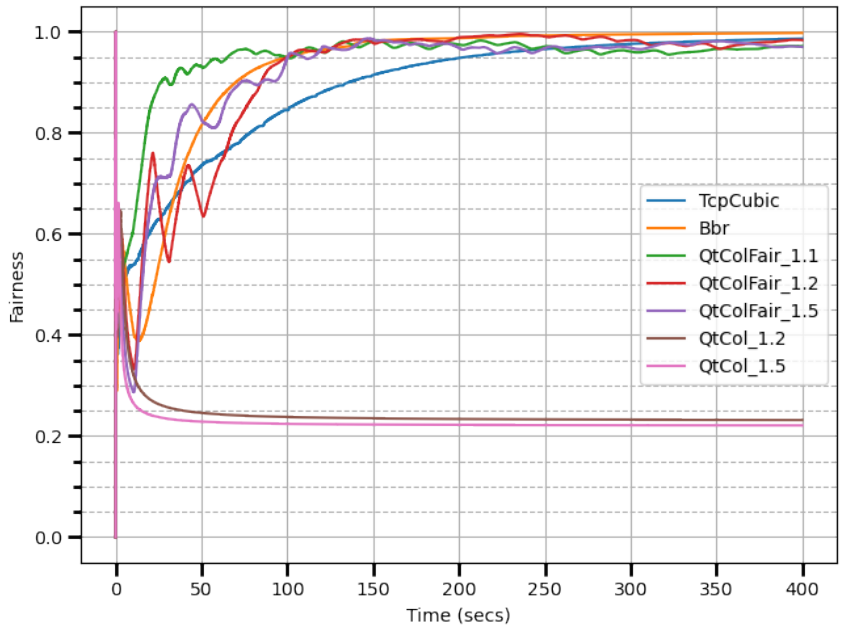

Figure 21. Jain's intra-fairness ratios in multiple-flow scenario.

TCP QtColFair promotes fairness by reducing the sending rate of high-throughput flows in favor of low-throughput flows, as shown in Figure 22. Conversely, TCP QtCol fails to ensure fairness, allowing a single flow to monopolize the available bandwidth, leaving other flows starved, as shown in Figure 23.

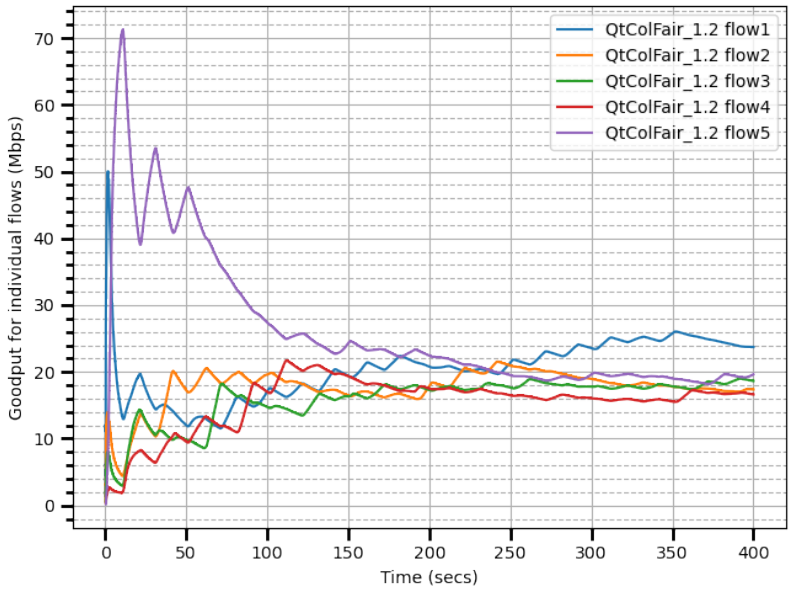

Figure 22. Convergence to fair sharing of available bandwidth by five TCP QtColFair flows.

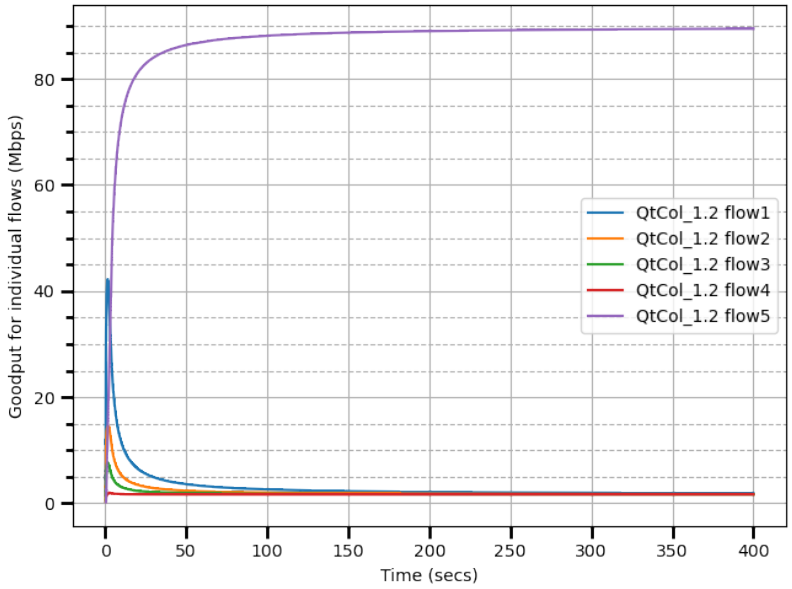

Figure 23. TCP QtCol's inability to share available bandwidth fairly.

5.5.4. Impact of Network Buffer Size

This section evaluates how varying network buffer sizes affect the performance of the congestion control algorithms. The BtlBW and PropDelay were fixed at 100 Mbps and 20 ms, respectively, while the buffer size was varied from 0.003 × BDP (near zero) to 10 × BDP.

Performance with Shallow Buffers ($\leq$1 × BDP):

As shown in Figure 24, TCP CUBIC performs slightly better in shallow buffers because packet loss and queueing delays are minimized. However, TCP BBR struggles due to its tendency to overestimate bandwidth, leading to high packet loss, as shown in Figure 25. TCP QtColFair responds to losses similarly to TCP NewReno, resulting in a moderate reduction in performance. As buffer size approaches 1 × BDP, TCP CUBIC and TCP QtColFair improve, but TCP BBR remains limited by packet losses.

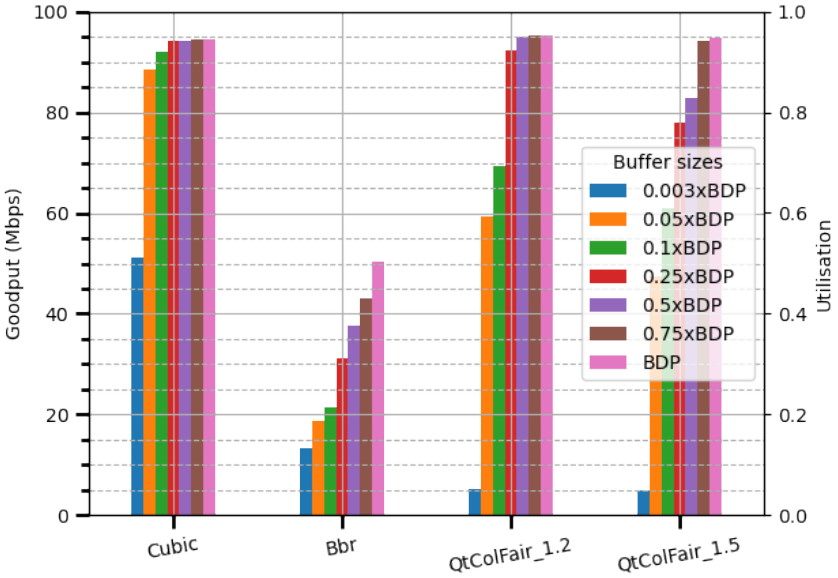

Figure 24. Goodput and effective network utilization in shallow network buffer size decreases below one BDP.

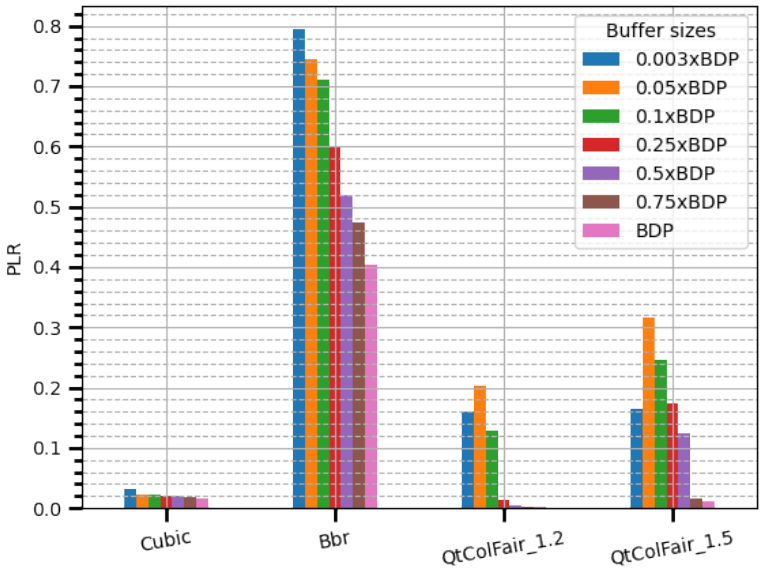

Figure 25. PLR comparison in shallow network buffers.

Performance with Deep Buffers:

As shown in Figure 26, TCP CUBIC's performance deteriorates with larger buffers due to increased queueing delays and higher packet loss in Figure 27. Both TCP BBR and TCP QtColFair improve as buffers grow. However, TCP QtColFair outperforms TCP BBR, achieving nearly 96% utilization with $\alpha = 1.2$, while TCP BBR remains below 95%. TCP QtColFair also performs better in avoiding queueing delays, as shown in the RTT comparison in Figure 28.

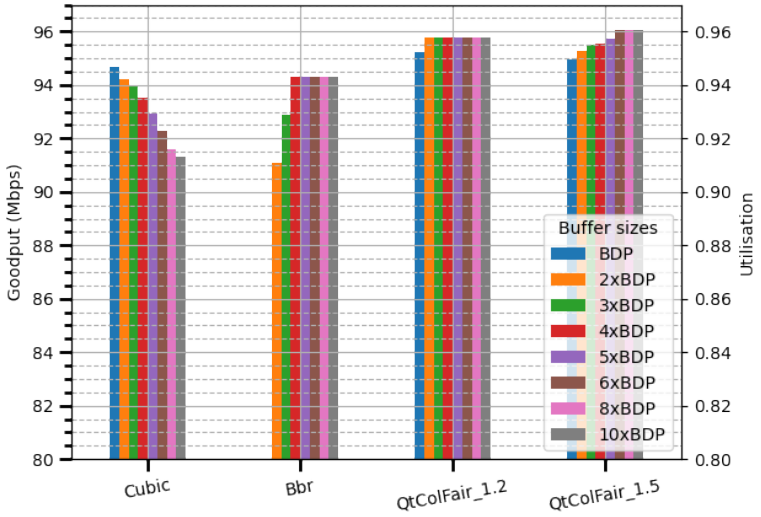

Figure 26. Goodput and effective network utilization as network buffer size increases above on BDP.

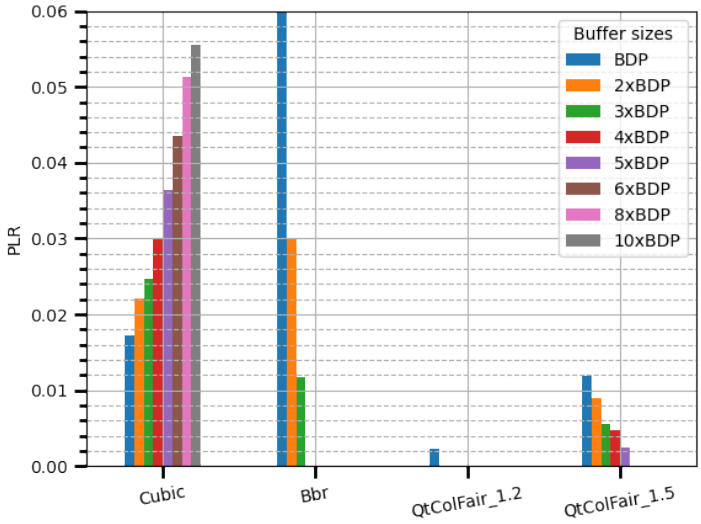

Figure 27. PLR comparison as network buffer becomes deeper.

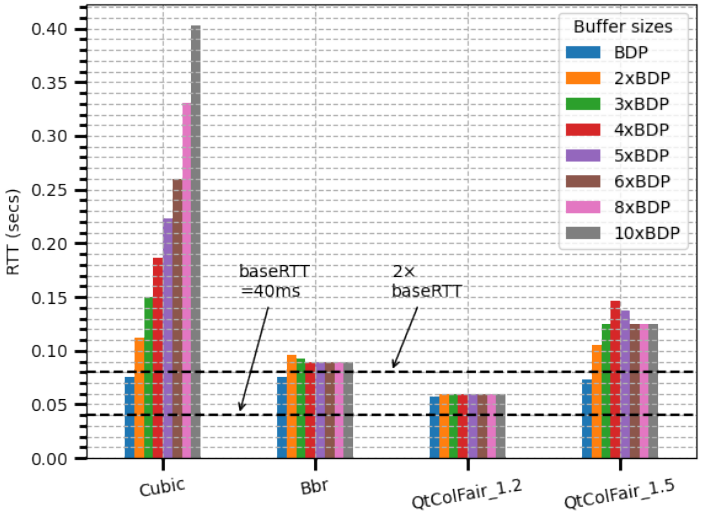

Figure 28. Average RTT comparison as network buffer becomes deeper. Values within the range of baseRTT and 2 × baseRTT are considered acceptable.

6. Conclusions

This article introduced TCP QtColFair, an innovative end-to-end TCP CCA designed to avoid queueing delays while optimizing data-in-flight in alignment with Kleinrock's optimality principle. As a delay-based algorithm, TCP QtColFair differentiates itself from existing approaches through the following key innovations:

- *Explicit Target RTT Specification:* The target RTT is defined as $\alpha \times$ minRTT, where α is fine-tuned to balance network utilization and congestion avoidance effectively.
- *Damping Mechanism Based on Harmonic Motion:* A novel damping framework solves an optimality equation to regulate the sending rate and data-in-flight smoothly over time, ensuring stability and efficiency.
- *Bandwidth Independence:* Unlike other Delay-Based algorithms, TCP QtColFair does not rely on bandwidth estimation, minimizing the impact of measurement inaccuracies and enhancing robustness.

The performance of TCP QtColFair was evaluated against TCP CUBIC (loss-based) and TCP BBR (delay-based) in multi-flow scenarios. Key findings include the following:

- *TCP CUBIC:* Exhibited large oscillations and data-in-flight values exceeding the bottleneck BDP and minRTT, leading to inefficient queue utilization.
- *TCP BBR:* Achieved Kleinrock's optimality on average but introduced oscillations due to its bandwidth and RTT probing mechanisms.
- *TCP QtColFair:* Consistently maintained data-in-flight at approximately $\alpha \times$ BDP and RTT near $\alpha \times$ minRTT, outperforming TCP BBR by avoiding queueing delays more effectively, particularly with smaller α values (e.g., $\alpha = 1.2$).

Additionally, TCP QtColFair demonstrated superior stability, eliminating oscillations over time in undisturbed conditions, unlike TCP BBR's bandwidth probing and TCP CUBIC's inherent oscillatory behavior. It achieved excellent goodput, reaching 96% utilization with a 5 $\times$ BDP buffer size, outperforming TCP BBR (94%) and TCP CUBIC (93%). In multi-flow scenarios, all algorithms exhibited fairness scores exceeding 0.9.

Future Work and Enhancements

Despite its strong performance, further improvements and research directions can refine TCP QtColFair and explore its potential in broader contexts.

Improvements needed:

- *Handling Packet Losses:* The current response to packet losses is akin to TCP NewReno, which can be overly aggressive. Future versions will enhance loss-handling mechanisms, particularly for networks with shallow buffers or high loss rates.
- *Improved RTT Refresh Mechanism:* A more robust change-point detection method is required to adjust minRTT dynamically, especially in multi-flow scenarios with significant RTT fluctuations. Preliminary studies indicate that TCP BBR also struggles in such conditions.

Further Research Directions:

- *Comparative Analysis:* Conduct detailed evaluations against newer versions of TCP BBR (e.g., BBRv2 and BBRv3) to benchmark performance under diverse network conditions.
- *Inter-Fairness and RTT Fairness:* Investigate fairness across different algorithms (inter-fairness) and among flows with varying round-trip times (RTT fairness).
- *Learning-Based Enhancements:* Explore the integration of the proposed mechanisms with machine learning algorithms. Machine learning can analyze global historical patterns to predict network behavior, while the proposed mechanism adapts in real time to dynamic network conditions.

- *Optimal Control Applications:* Study the application of classical optimal control theory to refine congestion control strategies and optimize system performance.

Further simulations and evaluations:

- *Topology-Based Analysis:* Evaluate performance in complex simulation environments, such as parking lot and randomized topologies, and in emerging architectures like 5G networks.
- *Stochastic Network Scenarios:* Assess performance under stochastic network conditions, incorporating random variations in bandwidth, delay, and packet loss rates.
- *Real-World Network Testing:* Validate performance in live network environments to ensure real-world feasibility and robustness.

Author Contributions: Conceptualization, D.W.N.; methodology, D.W.N., M.C.H. and B.T.M.; software, D.W.N.; validation, D.W.N., M.C.H. and B.T.M.; formal analysis, D.W.N.; investigation, D.W.N.; resources, D.W.N.; data curation, D.W.N.; writing—original draft preparation, D.W.N.; writing—review and editing, D.W.N., M.C.H. and B.T.M.; visualization, D.W.N., M.C.H.; supervision, B.T.M.; project administration, D.W.N.; funding acquisition, B.T.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The APC is funded by the SENTECH Chair in Broadband Wireless Multimedia Communications, University of Pretoria, https://www.up.ac.za/sentech-chair-in-broadband-wireless-multimedia-communication accessed on 4 November 2024.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available at https://github.com/dumisa/TowardsOptimalTcp accessed on 2 November 2024.

Acknowledgments: Would thank Sipho Khumalo, Moshe Masota, Mfanasibili Ngwenya and Phindile Ngwenya for support, reviews and comments.

Conflicts of Interest: The author D.W.N was employed by the company SENTECH SOC Limited. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations and definitions are used in this manuscript:

ACK, ACKs	Acknowledgement, acknowledgements.
AIMD	Additive-Increase, Multiplicative-Decrease.
baseRTT	Lowest RTT measured over entire TCP connection.
BDP	Bandwidth-delay-product, usually refers to network BDP given by minRTT $\times$ BtlBW. Also used as a unit of measure, e.g., for buffer size, data-in-flight, cwnd, etc.
BtlBW	Bottleneck bandWidth. Equivalent to μ.
CCA	Congestion control algorithm
cwnd	Congestion window. In a context it may refer to cwnd size or value.
ECN	Explicit congestion notification.
IP	Internet Protocol.
MIMD	Multiplicative-Increase, Multiplicative-Decrease.
minRTT	Lowest RTT observed over a specific time window.
ML	Machine Learning
MTU	Maximum Transmission Unit
PropDelay	One-way propagation delay.
RTT	Round-trip time. Equivalent to response time in a queueing system. While in real life RTT is not exactly twice one-way-delay, this article presumes RTT to be 2 $\times$ latency.
SCS	Successive constraint satisfaction.

TCP	Transmission control protocol.
TCP BBR	TCP BBR is one of the recent TCP CCAs based on Kleinrock's optimality condition for an optimal queuing system [24]. BBR stands for Bottleneck Bandwidth and Round-trip propagation time.
TCP CUBIC	A Loss-Based TCP CCA developed for high-speed networks [9]. It uses cubic increase function [32,33].
TCP NewReno	TCP NewReno is one of earliest Loss-Based TCP CCAs based on Jacobson's mechanism [32–34]. In most literature TCP CUBIC is used instead TCP Cubic even though CUBIC is not an abbreviation.
TCP QtCol	Basic implementation of the proposed mechanism or algorithm. Stands for TCP Queueing Theory Collocation—as reference to TCP congestion control using queueing theory and collocation methods.
TCP QtColFair	Practical implementation of the proposed mechanism or algorithm. Stands for TCP Queueing Theory Collocation Fair - as reference to TCP congestion control using queueing theory and collocation methods and incorporation of max-min fairness.
TCP Vegas	Earliest example of a Delay-Based TCP CCA [43].

The following symbols are used for equations in this manuscript:

B	Buffer size.
μ	network bandwidth capacity.
λ	arrival rate or aggregate sending rate.
ρ	Actual utilization.
x_s	sending rate for source s.
L; L_{predict}	Occupancy; Predicted value of L.
J	Jain's fairness ratio.
R; R_{target}; R_{min}	Response time or RTT; target RTT; minRTT or baseRTT.
α	Multiplicative factor for R_{target} from R_{min}.
W; W_{new}	cwnd; new cwnd.

Appendix A. Description of a Box-Plot

A box-plot (or box-and-whisker plot) provides a concise visual summary of data distribution, revealing its spread, skewness, and any potential outliers. It is particularly useful for comparing distributions across multiple datasets, and it has applications in congestion control research, such as in [15]. This structure offers a clear overview of data behavior, helping identify key statistical characteristics efficiently. An example of a box-plot is shown in Figure A1, which includes the following elements:

1. Box:
   - Rectangular box that captures the central spread of the data.
   - The bottom of the box represents the first quartile (Q1) or the 25th percentile.
   - The top of the box represents the third quartile (Q3) or the 75th percentile.
   - The height of the box (difference between Q3 and Q1) is called the interquartile range (IQR) and measures the spread of the middle 50% of the data.
2. Median:
   - A horizontal line inside the box marks the median (Q2 or 50th percentile).
   - The relative position of the median within the box hints at skewness: a shifted median suggests asymmetry.
3. Whiskers:
   - Extend from the box edges to capture data within 1.5 × IQR, about 99.2% for a normal distribution) from Q1 and Q3.
   - Show the range of the main data distribution, typically excluding extreme values.
4. Outliers:

- Data points lying beyond 1.5 × IQR from the quartiles, marked individually.
- Represent rare or extreme values that deviate from the general distribution (constitute less than 0.8% in a normal distribution). **Note:** No outliers are plotted in the example above.

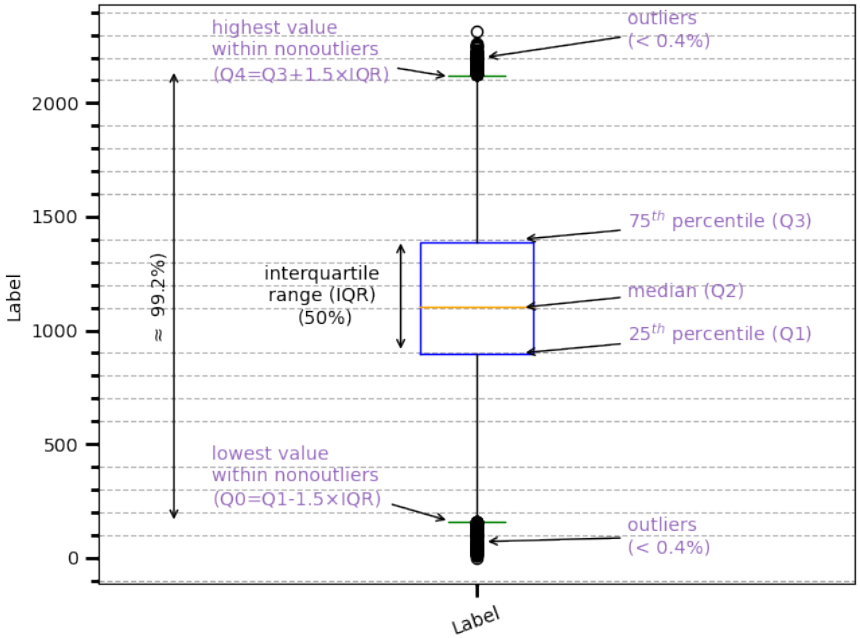

Figure A1. Illustration of a box-plot.

Appendix B. Modifications Made in ns-3.41 and Information for Conducting Simulations

1. The following files have been added in src/internet/model directory to implement TCP QtCol and TcpColFair, respectively:
   - tcp-qt-col.h and tcp-qt-col.cc for the basic implementation
   - tcp-qt-col-fair.h and tcp-qt-col-fair.cc for the practical implementation
2. The CMakeLists.txt file, in src/internet directory, is edited to add header and source files above for the build process.
3. my-dumbbell module is created in src/contrib directory with the files my-dumbbell.h and my-dumbbell.cc in src/contrib/my-dumbbell/model.
4. CMakeLists.txt is created in src/contrib/my-dumbbell with directives to build my-dumbbell module.
5. Several c++ and bash script files have been created for simulations in examples/tcp and sim_scripts directories, respectively.
6. Detailed information for simulations and python computation and visualization code are found at Github in https://github.com/dumisa/TowardsOptimalTcp [58] accessed on 2 November 2024.

References

1. Yuan, Y. Research on TCP Congestion Control Strategy Based on Proximal Policy Optimization. In Proceedings of the 2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 8–10 December 2023; Volume 11, pp. 1934–1938.
2. Verma, L.P.; Sharma, V.K.; Kumar, M.; Kanellopoulos, D. A novel Delay-based Adaptive Congestion Control TCP variant. *Comput. Electr. Eng.* **2022**, *101*, 108076. [CrossRef]
3. Varma, S. End-to-End versus Hop-by-Hop Congestion Control. In *Internet Congestion Control*; Romer, B., Ed.; Elsevier: Amsterdam, The Netherlands, 2015; p. 32.

4. Baker, F.; Fairhurst, G. *IETF Recommendations Regarding Active Queue Management*; RFC 7567; IETF: Wilmington, DC, USA, 2015.
5. Papadimitriou, D.; Zahariadis, T.; Martinez-Julia, P.; Papafili, I.; Morreale, V.; Torelli, F.; Sales, B.; Demeester, P. Design Principles for the Future Internet Architecture. In *The Future Internet: Future Internet Assembly 2012: From Promises to Reality*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7281, pp. 55–67.
6. Papadimitriou, D.; Welzl, M.; Scharf, M.; Briscoe, B. *Open Research Issues in Internet Congestion Control*; RFC 6077; IETF: Wilmington, DC, USA, 2011; pp. 1–51.
7. Lu, Y.; Ma, X.; Cui, C. DCCS: A dual congestion control signals based TCP for datacenter networks. *Comput. Netw.* **2024**, *247*, 110457. [CrossRef]
8. Mishra, A.; Sun, X.; Jain, A.; Pande, S.; Joshi, R.; Leong, B. The Great Internet TCP Congestion Control Census. In *SIGMETRICS '20, Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, Boston, MA, USA, 8–12 June 2020*; Association for Computing Machinery: New York, NY, USA, 2020; Volume 48, pp. 59–60.
9. Xu, L.; Ha, S.; Rhee, I.; Goel, V.; Eggert, L. *CUBIC for Fast and Long-Distance Networks*; RFC 9438; IETF: Wilmington, DC, USA, 2023.
10. Ha, S.; Rhee, I.; Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 64–74. [CrossRef]
11. Internet Engineering Task Force. IETF Homepage. Available online: https://www.ietf.org (accessed on 4 November 2024).
12. Alramli, O.I.; Hanapi, Z.M.; Othman, M.; Ahmad, I.; Samian, N. RTTV-TCP: Adaptive congestion control algorithm based on RTT variations for mmWave networks. *Ad Hoc Netw.* **2024**, *164*, 103611. [CrossRef]
13. Shrestha, S.K.; Pokhrel, S.R.; Kua, J. On the Fairness of Internet Congestion Control over WiFi with Deep Reinforcement Learning. *Future Internet* **2024**, *16*, 330. [CrossRef]
14. Naqvi, A.H.; Hilman, H.M.; Anggorojati, B. Implementability improvement of deep reinforcement learning based congestion control in cellular network. *Comput. Netw.* **2023**, *233*, 109874. [CrossRef]
15. Diel, G.; Miers, C.C.; Pillon, M.A.; Koslovski, G.P. RSCAT: Towards zero touch congestion control based on actor-critic reinforcement learning and software-defined networking. *J. Netw. Comput. Appl.* **2023**, *215*, 103639. [CrossRef]
16. Ma, S.; Jiang, J.; Wang, W.; Li, B. Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis. *arXiv* **2017**, arXiv:1706.09115.
17. Kleinrock, L. Internet congestion control using the power metric: Keep the pipe just full, but no fuller. *Ad Hoc Netw.* **2018**, *80*, 142–157. [CrossRef]
18. Al-Saadi, R.; Armitage, G.; But, J.; Branch, P. A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3609–3638. [CrossRef]
19. Zheng, S.; Liu, J.; Yan, X.; Xing, Z.; Di, X.; Qi, H. BBR-R: Improving BBR performance in multi-flow competition scenarios. *Comput. Netw.* **2024**, *254* , 110816. [CrossRef]
20. Jain, R. A Delay-Based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Comput. Commun. Rev.* **1989**, *19*, 56–71. [CrossRef]
21. Rodríguez-Pérez, M.; Herrería-Alonso, S.; Fernández-Veiga, M.; López-García, C. Common Problems in Delay-Based Congestion Control Algorithms: A Gallery of Solutions. *Eur. Trans. Telecommun.* **2011**, *22*, 168–178. [CrossRef]
22. Mittal, R.; Lam, V.T.; Dukkipati, N.; Blem, E.; Wassel, H.; Ghobadi, M.; Vahdat, A.; Wang, Y.; Wetherall, D.; Zats, D. TIMELY: RTT-based Congestion Control for the Datacenter. *Comput. Commun. Rev.* **2015**, *45*, 537–550. [CrossRef]
23. Cao, Y.; Jain, A.; Sharma, K.; Balasubramanian, A.; Gandhi, A. When to use and when not to use BBR: An empirical analysis and evaluation study. In *IMC '19, Proceedings of the ACM Internet Measurement Conference, Amsterdam, the Netherlands, 21–23 October 2019* ; Association for Computing Machinery: New York, NY, USA, 2019; pp. 130–136.
24. Cardwell, N.; Cheng, Y.; Gunn, S.C.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-Based Congestion Control. *Commun. ACM* **2017**, *60*, 58–66. [CrossRef]
25. Liao, X.; Tian, H.; Zeng, C.; Wan, X.; Chen, K. Astraea: Towards Fair and Efficient Learning-based Congestion Control. In *EuroSys '24, Proceedings of the 2024 European Conference on Computer Systems, Athens, Greece, 22–25 April 2024*; Association for Computing Machinery: New York, NY, USA, 2024; pp. 99–114.
26. Little, J.D.C. Little's law as viewed on its 50th anniversary. *Oper. Res.* **2011**, *59*, 536–549. [CrossRef]
27. Ngwenya, D.; Hlophe, M.C.; Maharaj, B.T. Towards Optimal End-to-end TCP Congestion Control Using Queueing-Based Dynamical Systems Theory. TechRxiv. 13 May 2024. Available online: https://www.techrxiv.org/doi/full/10.36227/techrxiv.171560562.26289531/v1 (accessed on 4 November 2024).
28. Lar, S.; Liao, X. An initiative for a classified bibliography on TCP/IP congestion control. *J. Netw. Comput. Appl.* **2013**, *36*, 126–133. [CrossRef]
29. Afanasyev, A.; Tilley, N.; Reiher, P.; Kleinrock, L. Host-to-host congestion control for TCP. *IEEE Commun. Surv. Tutor.* **2010**, *12*, 304–342. [CrossRef]

30. Lorincz, J.; Klarin, Z.; Ožegović, J. A Comprehensive Overview of TCP Congestion Control in 5G Networks: Research Challenges and Future Perspectives. *Sensors* **2021**, *21*, 4510. [CrossRef]
31. Bruhn, P.; Kühlewind, M.; Muehleisen, M. Performance and improvements of TCP CUBIC in low-delay cellular networks. *Comput. Netw.* **2023**, *224*, 109609. [CrossRef]
32. Jacobson, V. Congestion Avoidance and Control. *ACM SIGCOMM Comput. Commun. Rev.* **1988**, *18*, 314–329. [CrossRef]
33. Henderson, T.; Floyd, S.; Gurtov, A.; Nishida, Y. *The NewReno Modification to TCP's Fast Recovery Algorithm*; RFC 6582; IETF: Wilmington, DC, USA, 2012.
34. Allman, M.; Paxson, V.; Blanton, E. *TCP Congestion Control*; RFC 5681; IETF: Wilmington, DC, USA, 2009.
35. Arun, V.; Balakrishnan, H. Copa: Practical Delay-Based Congestion Control for the Internet. In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), Renton, WA, USA, 9–11 April 2018.
36. Tafa, Z.; Milutinovic, V. The Emerging Internet Congestion Control Paradigms. In Proceedings of the 2022 11th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 7–10 June 2022; pp. 7–10.
37. Boryło, P.; Biernacka, E.; Domżał, J.; Kądziołka, B.; Kantor, M.; Rusek, K.; Skała, M.; Wajda, K.; Wojcik, R.; Zabek, W. A tutorial on reinforcement learning in selected aspects of communications and networking. *Comput. Commun.* **2023**, *208*, 89–110. [CrossRef]
38. Jay, N.; Rotman, N.H.; Godfrey, P.B.; Schapira, M.; Tamar, A. A deep reinforcement learning perspective on internet congestion control. In Proceedings of the 36th International Conference on Machine Learning (ICML 2019), Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 5390–5399.
39. Zhang, L.; Cui, Y.; Wang, M.; Member, G.S.; Zhu, K. DeepCC: Bridging the Gap Between Congestion Control and Applications via Multiobjective Optimization. *IEEE/ACM Trans. Netw.* **2022**, *30*, 2274–2288. [CrossRef]
40. Piotrowska, A. Performance Evaluation of TCP BBRv3 in Networks with Multiple Round Trip Times. *Appl. Sci.* **2024**, *14*, 5053. [CrossRef]
41. Stidham, S.J. *Optimal Design of Queueing Systems*, 1st ed.; Chapman and Hall/CRC: New York, NY, USA, 2009.
42. Stidham, S.J. Optimal Control of Admission to a Queueing System. *IEEE Trans. Automat. Control* **1985**, *30*, 705–713. [CrossRef]
43. Brakmo, L.S.; Peterson, L.L. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE J. Sel. Areas Commun.* **1995**, *13*, 1465–1480. [CrossRef]
44. Varma, S. Analytic Modeling of Congestion Control. In *Internet Congestion Control*; Romer, B., Ed.; Elsevier: Amsterdam, The Netherlands, 2015; pp. 49–83.
45. Harchol-Balter, M. Queueing Theory Termilogy. In *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*; Cambridge University Press: Cambridge, UK, 2013; Chapter 2, pp. 13–26.
46. Dordal, P.L. *An Introduction to Computer Networks*; Loyola University Chicago: Chicago, IL, USA, 2018. Available online: https://intronetworks.cs.luc.edu (accessed on 4 November 2024).
47. Jain, R.; Chiu, D.; Hawe, W. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. arXiv 1984. Available online: https://arxiv.org/abs/cs/9809099 (accessed on 4 November 2024).
48. Dzivhani, M.; Ngwenya, D.; Masonta, M.; Ouahada, K. TCP congestion control macroscopic behaviour for combinations of source and router algorithms. In Proceedings of the 2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST), Accra, Ghana, 22–24 August 2018.
49. Geist, M.; Jaeger, B. Overview of TCP Congestion Control Algorithms. In *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)*; Chair of Network Architectures and Services: Garching, Germany, 2019; pp. 11–15. [CrossRef]
50. Chiu, D.; Jain, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.* **1989**, *17*, 1–14. [CrossRef]
51. Lahanas, A.; Tsaoussidis, V. Exploiting the efficiency and fairness potential of AIMD-based congestion avoidance and control. *Comput. Netw.* **2003**, *43*, 227–245. [CrossRef]
52. Kelly, F. Fairness and stability of end-to-end congestion. *Eur. J. Control* **2003**, *9*, 159–176. [CrossRef]
53. Gorinsky, S.; Georg, M.; Podlesny, M.; Jechlitschek, C. A Theory of Load Adjustments and its Implications for Congestion Control. *J. Internet Eng.* **2007**, *1*, 82–93.
54. Hock, M.; Bless, R.; Zitterbart, M. Experimental evaluation of BBR congestion control. In Proceedings of the 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017.
55. Pan, W.; Tan, H.; Li, X.; Xu, J.; Li, X. Improvement of BBRv2 Congestion Control Algorithm Based on Flow-aware ECN. *Secur. Commun. Netw.* **2022**, *2022*, 1218245. [CrossRef]
56. Biral, F.; Bertolazzi, E.; Bosetti, P. Notes on numerical methods for solving optimal control problems. *IEEJ J. Ind. Appl.* **2016**, *5*, 154–166. [CrossRef]
57. Rao, A.V. A Survey of Numerical Methods for Optimal Control. *Adv. Astronaut. Sci.* **2009**, *135*, 497–528.
58. Ngwenya, D. TCP QtCol and TCP QtColFair Implementation and Simulation [Source Code]. 2024. Available online: https://github.com/dumisa/TowardsOptimalTcp (accessed on 4 November 2024).

59. Borman, D. *TCP Options and Maximum Segment Size (MSS)*; RFC 6691; IETF: Wilmington, DC, USA, 2012.
60. Borman, D.; Braden, R.T.; Jacobson, V.; Scheffenegger, R. *TCP Extensions for High Performance*; RFC 7323; IETF: Wilmington, DC, USA, 2014.