future internet

MDPI

Article

On the Fairness of Internet Congestion Control over WiFi with Deep Reinforcement Learning

Shyam Kumar Shrestha, Shiva Raj Pokhrel * and Jonathan Kua

IoT & Software Engineering Research Laboratory, School of Information Technology, Deakin University, Geelong 3220, Australia; shyam.shrestha@deakin.edu.au (S.K.S.); jonathan.kua@deakin.edu.au (J.K.)
* Correspondence: shiva.pokhrel@deakin.edu.au

Abstract: For over forty years, TCP has been the main protocol for transporting data on the Internet. To improve congestion control algorithms (CCAs), delay bounding algorithms such as Vegas, FAST, BBR, PCC, and Copa have been developed. However, despite being designed to ensure fairness between data flows, these CCAs can still lead to unfairness and, in some cases, even cause data flow starvation in WiFi networks under certain conditions. We propose a new CCA switching solution that works with existing TCP and WiFi standards. This solution is offline and uses Deep Reinforcement Learning (DRL) trained on features such as noncongestive delay variations to predict and prevent extreme unfairness and starvation. Our DRL-driven approach allows for dynamic and efficient CCA switching. We have tested our design preliminarily in realistic datasets, ensuring that they support both fairness and efficiency over WiFi networks, which requires further investigation and extensive evaluation before online deployment.

Keywords: TCP unfairness; starvation; WiFi; dynamic CCA switching; congestion control algorithms (CCAs); Deep Reinforcement Learning

check for updates

Citation: 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. https://doi.org/10.3390/fi16090330

Academic Editor: Ping Wang

Received: 29 July 2024
Revised: 30 August 2024
Accepted: 3 September 2024
Published: 10 September 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Transmission Control Protocol (TCP) and the congestion control algorithms (CCAs) play a critical role in managing the flow of data across the Internet, directly impacting both Quality of Service (QoS) and Quality of Experience (QoE) for a wide range of applications [1,2]. Over the past several decades, researchers have focused on developing and refining these algorithms to ensure efficient and fair resource allocation at the transport layer, which is essential for maintaining high network performance and user experience/satisfaction [1]. CCAs govern the behavior of data transmission, particularly in response to packet loss, hence directly influencing key performance metrics such as throughput, end-to-end latency/Round Trip Time (RTT), jitter, and overall network stability.

Despite significant progress in recent years, users of Internet services continue to face substantial challenges, such as multi-second Request Completion Times (RCTs), which remains a problem [3]. This delay is primarily attributed to the limitations of existing delay-based CCAs, which adjust the congestion window (CWND) based on metrics such as RTT and estimated sending rates. While delay-based CCAs have been designed to enhance throughput and performance, particularly for interactive and real-time applications, their performance is often inconsistent and sub-optimal under varying network conditions.

1.1. Gaps in Existing Studies

Inefficiency of Loss-Based CCAs: Traditional loss-based CCAs, such as NewReno [4], CUBIC [5], and Compound TCP [6,7], increase their CWND in response to packet losses [8]. While these algorithms have been effective in maximizing throughput, they often result in increased latency and jitter, making them unsuitable for applications that require low latency, such as video conferencing and interactive online gaming. Moreover, these CCAs

are reactive by nature, responding to congestion events after they have occurred, which can lead to network inefficiencies and degraded user experience.

Limitations of Delay-Based CCAs: Delay-based CCAs, including Sprout [9], Remy [10], Vegas [11], FAST [12], BBR [13], PCC [14,15], Copa [16], and Verus [17], have been developed to address the limitations of loss-based algorithms by adjusting CWND based on delay measurements rather than packet losses. While promising, these algorithms are prone to performance degradation and fairness issues, particularly in networks where non-congestive delay variations—such as those caused by ACK aggregation, thread variations, or end-host scheduling—can significantly affect their operation. These variations can lead to situations where some flows receive disproportionately low bandwidth, a condition known as "starvation", which severely impacts QoE [18].

Fairness and Starvation Issues: Although delay-based CCAs are designed to improve network performance, they can inadvertently cause unfairness among competing flows, particularly under heterogeneous network conditions. For instance, if non-congestive delay variations exceed twice the equilibrium delay range, delay-based CCAs may become unstable, leading to starvation for certain flows. This issue is particularly problematic in WiFi networks, where varying signal strengths and interference can exacerbate these effects [19–21]. Existing studies often fail to adequately address these fairness issues, focusing instead on optimizing performance metrics like throughput or latency without considering the broader impact on user experience.

The Need for Dynamic Adaptation: Current CCAs are typically designed with a fixed logic that does not dynamically adapt to changing network conditions [22]. This rigidity is a significant shortcoming in modern networks, where conditions can vary rapidly, particularly in wireless and mobile environments [23]. The inability of these algorithms to adapt to real-time changes in network conditions leads to suboptimal performance [24] and exacerbates the fairness and efficiency issues described above. Moreover, the lack of dynamic adaptation makes it difficult for CCAs to maintain optimal performance across diverse environments, from stable wired connections to fluctuating wireless networks.

1.2. Why These Problems Need to Be Addressed

The challenges outlined above highlight the need for a more flexible and intelligent approach to congestion control. The persistent issues of unfairness, inefficiency, and starvation in existing CCAs undermine the potential of modern networks to deliver high QoS and QoE, particularly as applications become more demanding and network environments more complex.

Ensuring Fairness and Efficiency: Addressing the fairness and efficiency issues in CCAs is critical in providing a more equitable network experience. As users increasingly rely on real-time applications that require consistent and low-latency performance, the ability to ensure that all flows receive a fair share of network resources becomes essential [18,21]. Without solving these issues, certain applications and users will continue to suffer from poor performance, leading to widespread dissatisfaction [14] and potentially limiting the adoption of advanced Internet services.

Adapting to Diverse Network Conditions: The modern Internet is characterized by its diversity, with traffic patterns and network conditions varying widely across different environments. The static nature of existing CCAs makes them ill-suited to cope with this variability. Developing CCAs that can dynamically adapt to changing conditions in real time is crucial for maintaining optimal performance [24] and avoiding the pitfalls of traditional algorithms.

Leveraging Modern Technologies: The rise of machine learning and artificial intelligence presents an opportunity to develop more intelligent CCAs that can learn from network conditions and make real-time adjustments. By leveraging Deep Reinforcement Learning (DRL), it is possible to design CCAs that are not only more adaptive but also more capable of balancing the competing demands of throughput, latency, and fairness [19].

DRL-based approaches can predict and mitigate the risks of unfairness and starvation, leading to a more efficient and user-friendly Internet.

1.3. Proposed Solution and Contributions

To address these challenges, we propose a CCA switching mechanism that uses Deep Reinforcement Learning (DRL) to adjust congestion control strategies based on network conditions. This method is designed to work with existing TCP and WiFi standards, allowing the system to adapt to different network environments. Our DRL models are trained on features like noncongestive delay variations, helping to reduce the risk of unfairness and starvation in data transmission.

Current TCP CCAs, which rely on loss or delay, struggle to maintain fairness and efficiency under various network conditions. To improve this, we developed a DRL-based CCA switching mechanism that aims to make data transmission more equitable and efficient. Our key contributions are:

- **CCA Switching Mechanism:** We propose a DRL-based solution that adjusts congestion control strategies based on real-time network conditions, compatible with existing TCP and WiFi standards.
- **Delay Variation Analysis:** Our method uses DRL models trained on non-congestive delay variations to help avoid extreme unfairness and starvation.
- **Testing with Real Data:** We tested our approach offline over simulated environment using real data to ensure it aligns with the goals of fairness and efficiency in TCP. Online training and evaluation is beyond the scope of this paper.

In this paper, we first discuss prevalent TCP issues and the role of DRL in optimizing flow fairness, along with a review of relevant prior works in Section 2. Next, we present a comprehensive analysis of four widely used CCAs, mathematically demonstrating how unfairness and starvation occur in Section 3. In Section 4, we propose a DRL model to address these issues, detailing the model's design, training, and validation processes. Following this, Section 5 outlines our experimental setup, results, DRL model performance, and relevant graphs. Finally, we conclude by summarize our findings, discuss the limitations of our work, and provide directions for future work in Section 6.

2. Background and Related Works

Recently, several studies have shed light on fairness issues [21], including unfairness in TCP and extreme unfairness, also known as starvation. In [18], the authors presented experimental evidence demonstrating that achieving equal bandwidth distribution among network users is challenging. Instead, users often experience unfairness or a low share of available bandwidth, with at least one user receiving extremely low or zero bandwidth. This phenomenon, known as "TCP Starvation", is evident not only in delay-bounded congestion control algorithms but also in loss-based CCAs.

CCAs are crucial in network protocols, influencing data transmission performance and fairness. Traditionally, loss-based congestion control algorithms such as NewReno, CUBIC, and Compound TCP have been widely used. These algorithms adjust the congestion window size in response to packet loss, signaling network congestion. However, their reactive nature often leads to suboptimal performance as delay increases, posing challenges to maintaining fairness among flows [25].

In recent years, delay-bounding congestion control algorithms have garnered attention for their ability to control network traffic delay. Unlike their loss-based counterparts, delay-bounding algorithms prioritize delay as the primary congestion indicator. This approach is particularly advantageous for real-time and interactive applications requiring low latency. Notable delay-bounding algorithms include Vegas, FAST, BBR, PCC, and Copa, each with distinct design principles and mechanisms for delay control.

2.1. TCP Fairness and ML Approaches

Inter-flow fairness, ensuring equitable distribution of network resources among competing flows, is a fundamental aspect of congestion control. However, achieving fairness poses challenges, as highlighted by recent studies. These studies identify scenarios where starvation can occur despite efforts to maintain fairness, particularly in delay-bounding congestion control algorithms. Recently proposed solutions, including reinforcement learning-based approaches that aim to optimize host CCAs and bottleneck Active Queue Management (AQM) schemes [26], such as leveraging Deep Deterministic Policy Gradient (DDPG) [27] and Asynchronous Advantage Actor Critic (A3C) [28], aim to simultaneously improve throughput and ensure fairness [19].

Network delay variations significantly impact congestion control algorithm performance and fairness among flows. Factors contributing to delay variations challenge traditional throughput and fairness models. Comparative studies of TCP congestion control mechanisms emphasize the benefits of loss-based algorithms for latency-sensitive flows and the fairness issues of delay-based algorithms [29,30]. Empirical evidence highlights the potential for unfairness in delay-bounding congestion control algorithms, particularly when non-congestive network delay variations exceed certain thresholds. Improved fairness and performance have been achieved through adaptive and stable delay control algorithms. These advancements aim to address issues such as bufferbloat [31], which can lead to unfairness in network performance [32].

Metrics for fairness evaluation play a crucial role in assessing network protocols. Throughput fairness, delay fairness, jitter fairness, and loss fairness are key metrics used to evaluate the distribution of network resources among flows. These metrics help identify potential issues and guide the development of fairer congestion control algorithms [3].

ML and DRL frameworks for TCP offer promising avenues for improving congestion control algorithm fairness and performance. These frameworks enable the development of models that optimize network resource allocation while considering various QoE metrics. Integrating ML/DRL into congestion control algorithm selection processes can lead to more efficient and equitable network experiences [3,33,34].

Traditional rule-based approaches, which employ predefined rules and logical conditions, often struggle under varying network conditions due to their lack of flexibility. Heuristic methods, while relying on rule-of-thumb strategies and practical experience, can also fall short in complex and dynamic environments because they lack the ability to adapt to real-time changes [35]. Thus, instead of directly comparing DRL methods against these more rigid approaches, we chose to focus on a DRL-based dynamic switching mechanism. This approach leverages DRL's ability to continually learn and adapt to varying network conditions, offering a significant advantage over static evaluation functions or less flexible methods.

2.2. Closest Works in Literature

Arun et al. [18] highlighted that delay-bounding CCAs like Vegas, FAST, BBR, PCC, and Copa, while designed to ensure high network utilization and fairness, can lead to starvation under certain conditions, particularly when non-congestive delay variations exceed twice the equilibrium delay range. Their experiments with BBR, PCC Vivace, and Copa demonstrated that these CCAs may not effectively address inter-flow fairness in scenarios with significant delay variations, suggesting the need for CCAs to account for non-congestive jitter to prevent starvation. Similarly, Zhang et al. [3] addressed the gap between QoS optimization and actual application needs, proposing "Floo", a QoE-oriented mechanism that dynamically selects the most suitable CCA using reinforcement learning to improve web service performance by optimizing Request Completion Times.

Yamazaki et al. [36] reviewed the evolution of TCP variants, emphasizing the limitations of fixed-logic approaches in dynamic network environments and highlighting the potential of adaptive solutions like QTCP, a Q-learning-based TCP, despite its fairness issues. Zhang et al. [37] discussed the importance of congestion control mechanisms in maintaining Internet stability, noting the limitations of traditional loss-based algorithms

and the advancements in algorithms like BBRv2, Vivace, Copa, and C2TCP, which aim to balance high throughput with low delays. Finally, Xiao et al. [38] introduced "TCP-Drinc", a DRL-based approach that addresses the shortcomings of traditional TCP variants by adjusting the congestion window size using state features, demonstrating improved throughput and delay while maintaining fairness.

2.3. Deep Reinforcement Learning in TCP

Recent research has extensively explored the application of DRL and ML techniques to address fairness and performance issues in wireless networks and TCP congestion control.

In wireless networks, DRL-based approaches have demonstrated significant potential across various scenarios. For instance, DRL has been used to improve throughput and reduce collision rates in IEEE 802.11 networks by optimizing contention window parameters [39]. These techniques also enhance fairness and Quality of Service in wireless scheduling problems while maximizing system throughput [40]. In wireless mesh networks, a distributed network monitoring mechanism using Q-learning has been proposed to improve TCP fairness and throughput for starved flows [41]. Additionally, DRL-based MAC protocols have shown the ability to coexist with other MAC protocols while maximizing sum throughput or achieving proportional fairness, even without prior knowledge of the other protocols' operating principles [42]. These studies underscore the transformative potential of DRL in mitigating network unfairness and starvation issues across various wireless network scenarios.

Similarly, DRL and ML approaches have been applied to enhance fairness and efficiency in TCP congestion control. DRL techniques have improved fairness and performance in both single-flow and multi-flow scenarios [19,20]. These methods aim to optimize convergence properties such as fairness, fast convergence, and stability while maintaining high performance [20]. Researchers have also addressed the selfish behavior of learning-based congestion control algorithms by proposing new mechanisms that enhance fairness without compromising throughput and latency [36]. Furthermore, ML-based estimation of competing flows' congestion control algorithms has been explored to improve per-flow fairness in environments where multiple congestion control algorithms coexist [43]. Another DRL-based approach, TCP-Drinc [38], has been proposed to improve traditional TCP variants by adjusting the congestion window size using extracted state features and addressing challenges like delayed feedback and multi-agent competition, showing improved throughput and delay while maintaining fairness.

Moreover, RL-based approaches that aim to optimize host CCAs and bottleneck AQM schemes [26], such as leveraging Deep Deterministic Policy Gradient (DDPG) [27] and Asynchronous Advantage Actor-Critic (A3C) [28], have been proposed to simultaneously improve throughput and ensure fairness [19].

These studies highlight the potential of ML and DRL techniques to significantly enhance fairness and efficiency in TCP congestion control across diverse network environments. Existing DRL research has primarily focused on two approaches to enhance network performance, fairness, and to prevent starvation. The first approach involves optimizing traditional rule-based CCAs' functionality, such as the optimization of contention window parameters [38–40], convergence properties [20], and CCA and AQM [27,28]. The second approach involves designing DRL-based algorithms, such as TCP-Drinc [38], Aurora [44], and Astraea [20]. These efforts have mitigated many networking problems and reduced complexities.

However, implementing DRL in TCP is not straightforward. Some approaches are only available offline [20], require rigorous training for CCA, and demand high CPU resources, which are often lacking [45]. Additionally, existing approaches cannot guarantee fairness among competing flows as they are still in the development phase [46]. Despite these challenges, multiple studies have already demonstrated that DRL-based approaches are becoming increasingly promising in TCP.

Upon reviewing the existing ML-, RL-, and DRL-based approaches in TCP, we identified the need for a novel approach to further address persistent network issues. While

several DRL-based TCP protocols and optimization strategies have been proposed, no prior research has explored the dynamic switching between available CCAs to enhance network performance, improve flow fairness, and prevent starvation, especially in heterogeneous environments, such as in WiFi networks. This paper introduces a DRL-based dynamic switching mechanism designed to tackle these persistent TCP issues. Our results demonstrate significant improvements in key performance metrics such as throughput, RTT, jitter, packet loss, fairness, and the prevention of throughput starvation.

Despite advancements in congestion control algorithms, several challenges remain unresolved. A comprehensive understanding of how delay-bounding algorithms adapt to non-congestive network delay variations, the thresholds and characteristics of non-congestive jitter, and the development of ML/DRL frameworks to quantify and mitigate fairness issues is essential. Addressing these gaps will advance the development of congestion control algorithms and contribute to enhanced network performance and fairness [19].

In this study, we focused on TCP unfairness by analyzing the underlying mathematical principles, conducting real-world network tests, and dynamically switching CCAs using DRL techniques. This involved assessing current CCAs' performance under varying network conditions and dynamically selecting the optimal CCA when significant disparities in throughput, latency, loss rate, and sending rate were observed among competing flows. To the best of our knowledge, this is the first study to investigate the dynamic switching of CCAs using DRL to improve flow performance and prevent starvation. Therefore, this research aims to contribute to the field by providing a DRL-based approach that dynamically switches CCAs based on historical network data to better understand and respond to network fluctuations, ultimately improving flow performance and preventing starvation.

3. Mathematical Interpretation of Unfairness and Starvation in Representative CCAs

Multiple TCP CCAs exist, but all have the common core philosophy of improving users' experience in reliable data transformation. Some of the CCAs explicitly in use are Bottleneck Bandwidth and Round-trip propagation time (BBR), Performance-oriented Congestion Control (PCC) Vivace, Vegas, Copa, CUBIC, NewReno, and Reno. In this section, we aim to provide a detailed discussion of four key CCAs: BBR, PCC Vivace, CUBIC, and Copa. The exposition will elucidate the comprehensive functioning of each TCP variant, shedding light on their overall behavior and how they respond to packet loss and variations in RTT across flows.

Before moving to the detailed mathematical interpretation, we briefly explain the hypotheses formulated later in this section. Three key hypotheses are made for each selected CCA:

- *Equal Loss with Varying RTT:* This hypothesis aims to investigate how a CCA manages flows with different RTTs when they experience the same level of packet loss. Specifically, it evaluates whether disparities in RTTs result in unequal congestion window allocation under identical loss conditions. The primary purpose is to ascertain whether variations in RTT alone can lead to unfairness in congestion window sizes across different flows.
- *Equal RTT with Varying Loss:* In this hypothesis, the focus is on how varying levels of packet loss impact the congestion window of flows with the same RTT. It examines whether changes in packet loss lead to fairness issues or potential starvation in flows that experience identical network delays. The objective is to investigate whether packet loss alone can disrupt fairness and lead to differential treatment of flows.
- *Starvation Hypothesis:* This hypothesis examines extreme cases of unfairness, where one flow significantly underperforms compared to another due to variations in RTT or packet loss. The aim is to assess the worst-case scenarios for fairness and evaluate how effectively the CCA prevents starvation.

"TCP Starvation" [18] is a critical phenomenon that can significantly impact network performance and fairness. It occurs when a TCP flow with a longer RTT receives a dispro-

portionately smaller share of bandwidth/observed ratio compared to flows with shorter RTT. This can lead to degraded throughput, increased latency, and unfair resource allocation.

TCP CUBIC performs the window growth functions in a cubic function. It sets a Maximum Window ($W_{\max}$) at the point where the packet loss occurred. Then, the congestion window decreases multiplicatively. It recovers the performance quickly and enters the congestion-avoidance phase. Up to the inflection point, the window size increases in concave style. However, it adjusts the window size after the congestion is detected and maintains stabilized networks at $W_{\max}$ before entering convex-style growth of the window size. Concave and convex styles of window adjustments assist CUBIC to stand out from the other existing congestion control algorithms, improving protocol and network stability by maintaining high network utilization. Improvements are possible as the window size remains stable around $W_{\max}$ by forming a plateau. The detailed derivation are provided at the end of this paper. With extensive analytic modeling, we consider two cases to explain the causes of unfairness and starvation.

CASE I: Considering two CUBIC flows with different RTTs, τ_1 and τ_2, and experiencing the same loss p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{\tau_1^3}{\tau_2^3}} \tag{1}$$

using (1); if $\tau_1 = 2\tau_2$, then $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{8}$, which explain the key reason for the observed unfairness.

CASE II: Considering two CUBIC flows with the same RTTs, τ, and experiencing different losses, p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve:

$$\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{p_2^3}{p_1^3}} \tag{2}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{1}{8}}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{1}{n^3}}$;

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{n^3}$.

PCC Vivace borrows the PCC architecture: a utility function framework and a learning rate-control algorithm and perceives both modules uniquely. This utility function rewards throughput to maximize performance and minimize latency, while it penalizes when packet loss occurs and increases latency [14,15]. The learning rate-control algorithm provides an opportunity for the sender to select the sending rates that allow the senders to learn their performance through statistic aggregation, such as achieved throughput, latency, and loss rate of packets. Then based on the numerical utility value, they determine the sending rate [15]. The following equation helps to find out the utility value. With detailed derivation deferred to the Appendix A, we consider two cases to explain the causes of unfairness and starvation.

CASE I: Considering two PCC flows with different RTTs, τ_1 and τ_2, and experiencing the same loss p, we can evaluate how the congestion windows $U_1^\star$ and $U_2^\star$ evolve as

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - b\frac{d\tau_1}{dT} - cL_i}{1 - b\frac{d\tau_2}{dT} - cL_i} \tag{3}$$

If $\tau_1 = 2\tau_2$, then $\frac{U_1^\star}{U_2^\star} = \frac{1-2b\frac{d\tau_2}{dT} - cL_i}{1-b\frac{d\tau_2}{dT} - cL_i}$.

CASE II: Considering two PCC flows with same RTTs, τ, and experiencing different losses, p_1 and p_2, we can evaluate how the congestion windows $U_1^\star$ and $U_2^\star$ evolve:

$$\frac{U_1^\star}{U_2^\star} = \frac{(1 - b\frac{d\tau}{dT} - cL_i)(1 - p_2)}{(1 - b\frac{d\tau}{dT} - cL_i)(1 - p_1)} = \frac{1 - p_2}{1 - p_1} \tag{4}$$

If $p_1 = 2p_2$, then $\frac{U_1^\star}{U_2^\star} = \frac{1-p_2}{1-2p_2}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = \{1, 2, 3, \ldots, 20\}$

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - p_2}{1 - np_2}$$

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, 2, 3, \ldots, 20$

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - n(b\frac{d\tau_2}{dT}) - cL_i}{1 - (b\frac{d\tau_2}{dT}) - cL_i}$$

BBR has a strong influence on congestion control, which measures the available Bottleneck Bandwidth (Btlbw) and lowest RTT [47]. By using those measurements, BBR creates a network path model to increase the delivery rate and decrease latency [47]. The BBR congestion control algorithm also measures the maximum delivery rate and minimum transmission delay to find Kleinrock's optimal operating point [48], also known as Bandwidth Delay Product (BDP). During this period, the delivery rate remains unchanged but RTT increases. With extensive mathematical modeling detailed at the end of this paper, we consider cases to explain the causes of unfairness and starvation as follows.

CASE I: Considering two BBR flows with different RTTs, τ_1 and τ_2, and experiencing the same loss p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau_1}{\tau_2}\left(\frac{\tau_2 - 2\tau_m}{\tau_1 - 2\tau_m}\right) \tag{5}$$

If $\tau_1 = 2\tau_2$, then

$$\frac{W_1^\star}{W_2^\star} = \frac{2\tau_2}{\tau_2}\left(\frac{\tau_2 - 2\tau_m}{2\tau_2 - 2\tau_m}\right)$$

CASE II: Considering BBR flows with the same RTTs, τ, and experiencing different losses p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau - 2\tau_m(1 - p_2)}{\tau - 2\tau_m(1 - p_1)} \tag{6}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-2p_2}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-np_2}$

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{3n}{5n-2}$

Copa is a delay-based end-to-end congestion control mechanism. It observes the delay evolution to detect the existing buffer fillers and reacts with an additive increase/multiplicative decrease [16]. While the detailed maths are deferred to the end of this paper, we obtain the equilibrium congestion window $W^\star$ as

$$W^\star = \frac{\tau^\star}{\delta(\tau^\star - T_m)} \tag{7}$$

We consider two cases to explain the causes of unfairness and starvation.

CASE I: Considering two Copa flows with different RTTs, τ_1 and τ_2, and experiencing the same loss, p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau_1}{\tau_2}\left(\frac{\tau_2 - T_m}{\tau_1 - T_m}\right) \tag{8}$$

If $\tau_1 = 2\tau_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{2\tau_2}{\tau_2}\left(\frac{\tau_2 - T_m}{2\tau_2 - T_m}\right)$.

CASE II: Considering Copa flows with same RTTs, τ, and experiencing different losses p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve:

$$\frac{W_1^\star}{W_2^\star} = \frac{(\tau - T_m)(1 - p_2)}{(\tau - T_m)(1 - p_1)} = \frac{1 - p_2}{1 - p_1} \tag{9}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-2p_2}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-np_2}$

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, 2, 3, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{n\tau_2}{\tau_2}\left(\frac{\tau_2 - T_m}{n\tau_2 - T_m}\right)$

Figure 1a highlights the possibilities of flow starvation under different TCP variants and congestion levels. An n represents the loss ratio, whereas the ratio observed by TCP flows reflects the relative bandwidth allocation among the various flows of the PCC Vivace, BBR, Copa, and CUBIC. Since n represents the loss ratio, higher n values mean more packet loss, which indicates the intensity of congestion in the network. Each pair of color-coded lines corresponds to two flows using the same TCP variants and highlights how the bandwidth ratio between those flows varies as the loss ratio increases.

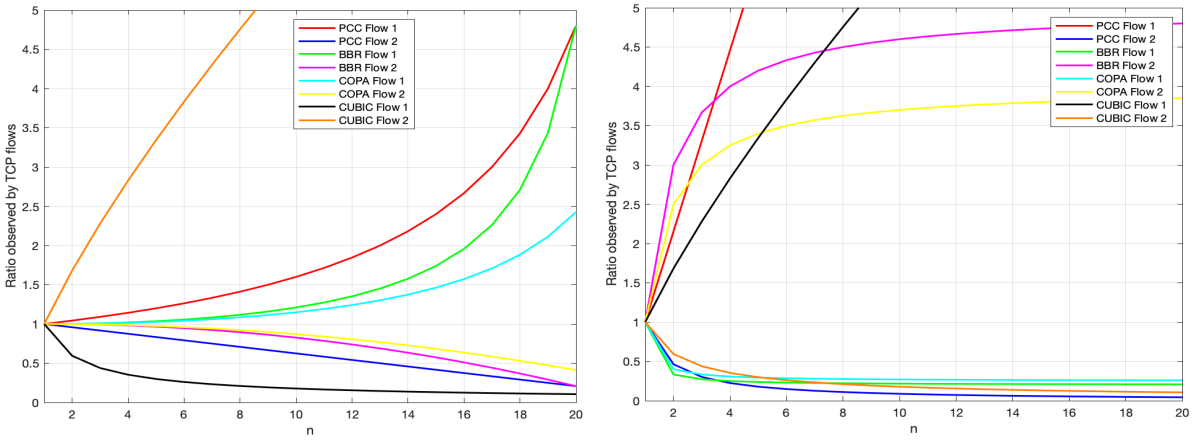


Figure 1. Starvation in multiple TCP protocols (PCC Vivace, BBR, Copa, and CUBIC) with (**a**) different loss ratio of 2 flows ($p_1 = np_2, n =$ loss ratio $= 1, \ldots, 20$), where p_1 and p_2 represent flow 1 and flow 2 loss ratios, CASE I: Starvation from Table 1 with different loss ratios, $n = 1, 2, \ldots$: (**b**) ($\tau_1 = n\tau_2, n =$ RTT ratio $= 1, \ldots, 20$) , where τ_1 and τ_2 represents flow 1 and flow 2 RTT ratios, CASE II: Starvation from Table 1 with different RTT ratios, $n = 1, 2, \ldots$.

Table 1. Summary of congestion control for CUBIC, BBR, PCC Vivace, and Copa.

Algorithms	Window $\star(W^{\star})$	Unfairness	Starvation
CUBIC [49]	$\sqrt[4]{\frac{\tau^3 C}{(p^{\star})^3 \beta}}$	$\sqrt[4]{\frac{\tau_1^3}{\tau_2^3}}, \sqrt[4]{\frac{p_2^3}{p_1^3}}$	$\sqrt[4]{n^3}, \sqrt[4]{\frac{1}{n^3}}$
BBR [50]	$\frac{\alpha\tau^{\star}}{\tau^{\star}-2\tau_m}$	$\frac{\tau_1}{\tau_2}\left(\frac{\tau_2-2\tau_m}{\tau_1-2\tau_m}\right), \frac{\tau-2\tau_m(1-p_2)}{\tau-2\tau_m(1-p_1)}$	$\frac{1-p_2}{1-np_2}, \frac{3n}{5n-2}$
PCC [14,15]	$1-b\frac{d\tau_i}{dT}-cL_i$	$\frac{1-b\frac{d\tau_1}{dT}-cL_i}{1-b\frac{d\tau_2}{dT}-cL_i}, \frac{1-p_2}{1-2p_2}$	$\frac{1-p_2}{1-np_2}, \frac{1-n(b\frac{d\tau_2}{dT})-cL_i}{1-(b\frac{d\tau_2}{dT})-cL_i}$
Copa [16]	$\frac{\tau^{\star}}{\delta(\tau^{\star}-T_m)}$	$\frac{\tau_1}{\tau_2}\left(\frac{\tau_2-T_m}{\tau_1-T_m}\right), \frac{1-p_2}{1-p_1}$	$\frac{1-p_2}{1-np_2}, \frac{n\tau_2}{\tau_2}\left(\frac{\tau_2-T_m}{n\tau_2-T_m}\right)$

In Figure 1b, the RTT Ratio (n) quantifies the difference in Round-Trip Times between competing flows, with higher values pointing to greater disparities. As the n increases, lines for flows with longer RTT curves move downwards, indicating declining bandwidth ratios. The ratio observed by TCP flows measures bandwidth distribution among flows, and values falling significantly below 1 suggest starvation for the corresponding flows. Notably, different TCP protocols, such as PCC, BBR, Copa, and CUBIC, demonstrate distinct behaviors concerning starvation.

Based on our analysis of plots and equations associated with different TCP algorithms, our hypothesis regarding resource allocation disparity seems to hold true. We initially hypothesized that regardless of design principles aiming for TCP-friendliness, low latency, low loss, and higher throughput, fairness issues related to resource allocation still arise. Our observations within a specific congested network environment suggest that packets experience significant queuing delays, leading to delivery time extensions or packet drops, even when certain flows manage to deliver packets without issues. This scenario represents a "starvation state" for some flows, and such intra- and inter-variant unfairness negatively impacts overall network performance.

4. System Model of Proposed TCP-Switching Mechanism

This section details the design of our TCP-switching mechanism, which utilizes DRL to optimize CCAs. The model integrates four key components, Unfairness Monitoring, Collective Statistics, DRL-based Selection, and TCP Switching, each contributing to a dynamic and responsive network management system.

Before presenting the details of our system model, we outline a set of metrics in Tables 2 and 3 for monitoring unfairness and Collective Statistics in CCAs, respectively. Analyzing these metrics allows us to assess CCA fairness and identify the need for adjustments or switching.

Table 2. Metrics to monitor the unfairness of CCAs.

Metric	Interpretation
Th_{ratio}	Throughput of flow A/Throughput of flow B
τ	Travel time from source to destination and back to source
Delay	One way delay—from source to the destination
P_{rate}	Number of loss packets/Total number of sent packets
Jitter	Variability in packet arrival times
Goodput	(Total transferred data—overhead data)/time taken
Sq_{size}	Total amount of buffered data waiting to be transmitted
$byte_{rate}$	Bytes the sender processes in the last selection period

Table 3. Metrics accumulated by Collective Statistics.

Metric	Interpretation
Th_{max}	The optimum delivery rate
τ_{min}	The minimum Round Trip Time
τ_{rate}	Average τ/τ_{min}
P_{rate}	Number of lost packets/Total number of sent packets
CCA_{name}	Running CCA
Goodput	(total transferred data—overhead data)/time taken
Sq_{size}	Total amount of buffered data waiting to be transmitted
$byte_{rate}$	Bytes the sender processes in the last selection period

Selecting effective CCAs requires addressing flow unfairness, which is pivotal for optimizing QoS and QoE. We assert that CCAs failing to distribute bandwidth fairly among all flows will not adequately enhance QoS and QoE. Our approach involves evaluating and addressing the fairness of CCAs through various metrics and employing DRL to switch CCAs based on these evaluations dynamically.

To detect unfairness in CCAs, we use a fixed-point approach that compares throughput across different flows. If one flow consistently receives a higher throughput than others, it indicates that the CCA may be unfair. This approach involves defining a fairness function that evaluates the throughput ratio between flows and iteratively adjusting it until it reaches a fixed point, reflecting an equitable bandwidth distribution. For example, one possible function could be the ratio of the throughput of the flow to the average throughput of all flows:

$$f(Th_1) = \frac{Th_1}{a.Th_{avg}}$$

where Th_1 is the throughput of flow 1, a is the number of flows, and Th_{avg} is the average throughput of all flows.

The fixed-point approach then iteratively adjusts the throughput of each flow until the function $f(Th_1)$ reaches a fixed point. A fixed point is a value of the function that does not change when the throughput of the flows is adjusted. The fixed-point approach can be shown in the following equation:

$$Th_1(x+1) = f(Th_1(x))$$

where $Th_1(x)$ is the throughput of flow 1 in iteration x, $Th_1(x+1)$ is the throughput of flow 1 in iteration $x+1$, and $f(Th_1(x))$ is the value of the function $f(Th_1)$ in iteration x. If the fixed-point approach converges to a value where all flows have the same throughput, then the congestion control algorithm is fair. However, if the fixed-point approach converges to a value where one flow has more throughput than the others, then the congestion control algorithm is unfair.

CCAs are deployed in the transport layer that settles behaviors of data transformation, which rigorously affects the user experience [3]. The improvisation in QoS does not improve the Quality of Experience. The gap between QoE and QoS must be filled; if not, it must be minimized. Here, we do not upgrade any CCAs directly; however, we attempt to develop a model between the transport layer and the application layer that assists in selecting the right CCAs and ultimately provides better QoE. There is no such CCA that fits in every situation and functions perfectly in all scenarios. Thus, the alternative for better QoE is to autonomously select the effective and efficient CCA in varying scenarios.

However, it is challenging to select the right CCAs and to form the CCA-selection policies [3] by observing the network conditions and QoE metrics. One of the main challenges is to adapt to the dynamic network conditions. The networks are affected by various factors [3] such as the overflow of traffic into the networks, the fading of wireless channels, and user movement activities. The network's condition does not stay stable. Another difficulty in choosing the right CCA is the empirical characteristics of CCAs [51,52].

The latest CCAs [14,16] are seriously difficult to model and characterize, as the existing understanding of CCAs is empirical [51,52].

Another notable challenge is the difficulty in smooth switching. As [3] mentioned, there is a very strong possibility of taking the place of CCA switching while the transmission is happening because of unstable network conditions. Here, we have implemented unrestricted switching. Unrestricted switching allows the new CCA to inherit all CCA-related variables, encompassing connection-level variables (such as congestion window/sending rate and RTT-related values), and private state variables of the CCA (e.g., minimum RTT, CWND increment, etc.). Limited sSwitching, on the other hand, only inherits connection-level variables. The private state variables of the new CCA are initialized from default values.

4.1. Model Interpretation

As shown in Figure 2, Collective Statistics aggregates data from the transport and application layers. These metrics are integrated with inputs from unfairness monitoring. The DRL-based selection then evaluates the fairness status of CCAs using these data, maps the metrics provided by Collective Statistics, selects an optimal CCA, and recommends TCP switching. Upon receiving the optimal CCA recommendation, TCP switching initiates the transition from the current CCA to the newly selected optimal CCA, considering the network dynamism and environments.

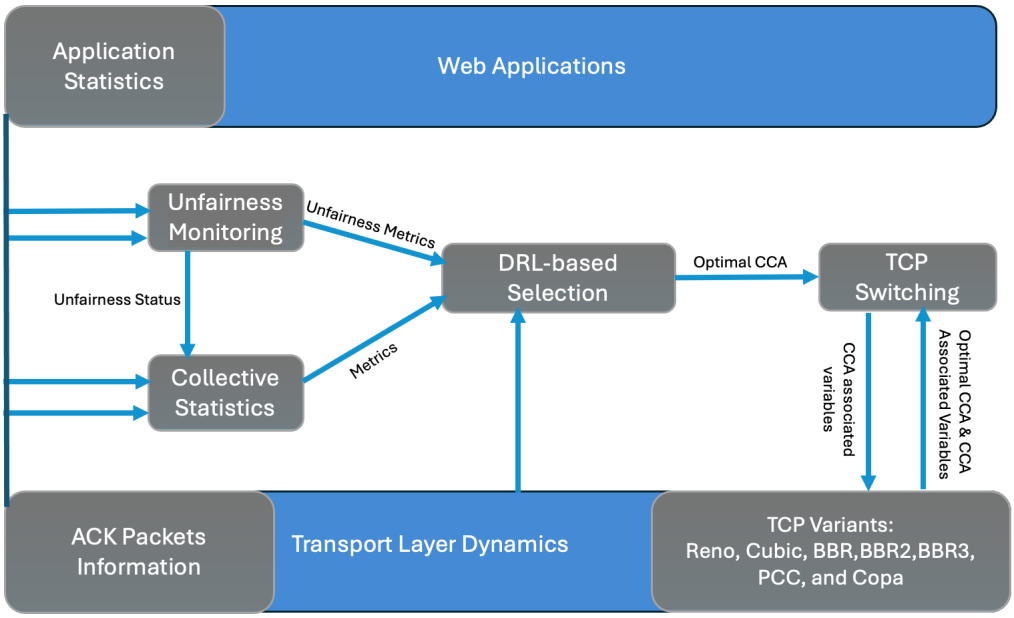


Figure 2. A high-level view of our model design with different components and their interactions.

4.1.1. Proposed DRL-Based CCA Switching

Our DRL framework utilizes a Deep Q-Network (DQN) to dynamically select and switch CCAs based on real-time network conditions. DQN was chosen over other DRL algorithms, such as DDPG, Double DQN, and DeepSARSA due to its superior performance in fairness analysis, as highlighted by [53], and its effectiveness in handling large discrete action spaces, which is relevant for CCA selection [54]. Furthermore, [55] demonstrated the ability of DQN to manage complex decision-making tasks, further supporting its suitability for our research.

State Space (S)

The state space captures key network metrics influencing the performance of CCA, including throughput, RTT, loss rate, and jitter. These parameters allow the DQN to assess and adjust the CCA based on real-time conditions.

Action Space (A)

The action space consists of selecting between different CCAs, such as *Cubic*, *BBR*, or *PCC*, where each action represents a potential CCA switch.

Reward Function (R)

The reward function is designed to improve network performance by addressing disparities in metrics and promoting beneficial CCA switches. For simplicity, we use

$$R(s_t, a_t) = \frac{1}{2}\log(\text{Throughput}(t)) - \frac{1}{4}\log(\text{Latency}(t)) - \frac{1}{4}\log(\text{LossRate}(t)),$$

but the constants can be tuned to encourage actions that emphasize throughput and reduce RTT and the loss rate differently.

Q-Function (Q)

The Q function estimates the expected cumulative reward for each pair of state actions, which guides the selection of the optimal CCA:

$$Q(s_t, a_t) = \mathbb{E}\left[R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')\right]$$

with $\gamma = 0.95$ balancing immediate and future rewards. This function helps determine actions that maximize long-term network performance. DRL trains a neural network to solve problems to optimize our proposed approach, predicting and responding to congestion indicators, effectively using the available bandwidth, and switching to the optimal CCA. Due to current dependencies on user-space libraries, DRL implementations must run in the user space (not in the kernel). The proposed DRL-Based TCP Switching Algorithm is shown in Algorithm 1.

Given time and resource constraints, network environment complexity, data preprocessing needs, and real-time modeling challenges, we implemented offline DRL as the sole method for switching CCAs in this paper. The process was as follows.

Data Collection: Historical network traffic data were gathered using tools like iperf3 and Wireshark, capturing metrics such as throughput, packet loss, RTT, and congestion levels.

Offline Training: The DRL model was trained offline with the collected data, learning to identify network behavior patterns and select suitable CCAs. Validation: The model was validated with separate datasets to ensure its effectiveness and generalization to unseen network conditions, fine-tuning parameters as needed.

Validation The DRL model was validated with separate datasets to ensure its effectiveness and generalization to unseen network conditions, fine-tuning parameters as needed.

Realtime Monitoring: The continuous monitoring of real-time network metrics was conducted to update network condition variables. These metrics were inputted into the trained DRL model to predict the optimal CCA for current network conditions.

Algorithm Selection Policy: A policy was developed based on the DRL model's decisions, determining CCA selection based on observed network conditions.

Simulation: The DRL-based congestion control mechanism was simulated in an offline environment, evaluating performance and the ability to mitigate unfairness.

Optimization: The DRL model and selection policy were refined based on simulation results and validation feedback, enhancing decision-making accuracy and robustness.

Preparation for Switching: The parameters of the predicted optimal CCA were retrieved, potential disruptions during the transition were analyzed, and the current state of network traffic was saved to facilitate a smooth migration to the new CCA.

Execution of Switching: New CCA parameters were communicated to the transport layer, variables were migrated smoothly to minimize disruption, and the new CCA was applied while monitoring initial performance.

Algorithm 1 DRL-Based TCP Switching Algorithm

Input: Historical network data, Real-time network metrics
Output: CCA selection and smooth switching
Step 1: Data Collection
Collect historical network traffic data using tools (e.g., iperf3, Wireshark)
Gather metrics: throughput, packet loss, RTT, congestion levels
Step 2: Offline Training
Train DRL model with collected historical data
Model learns to identify patterns and select suitable CCA
DRL model analyzes historical network metrics (e.g., throughput, packet loss, RTT, congestion levels)
the model identifies patterns and correlations within these metrics that impact network performance
The model learns to predict the best-performing CCA under a different network
Select the optimal perfroming CCA
Step 3: Validation
Validate the DRL model with separate validation datasets
Fine-tune model parameters to improve generalization
Step 4: Real-time Monitoring
while network is operational **do**
    Continuously monitor real-time network metrics
    Update network condition variables
    Input real-time metrics to the trained DRL model
    Predict the optimal CCA for current network conditions
end while
Step 5: Algorithm Selection Policy
Develop policy based on the DRL model's decision-making
Determine appropriate CCA based on observed conditions
Step 6: Prepare for Switching
Retrieve parameters of the predicted optimal CCA
Analyze potential disruptions during the transition
Save the current state of network traffic
Step 7: Execute Switching
Communicate new CCA parameters to the transport layer
Migrate CCA variables smoothly, ensuring minimal disruption
Apply the new CCA and monitor initial performance
Step 8: Post-Switch Evaluation
Continue monitoring network metrics post-switch
Compare performance with pre-switch metrics
if performance improves **then**
    Confirm new CCA selection
else
    Revert to the previous CCA or select an alternative
end if

5. Experimental Methodology, Results, and Discussion

In this section, we present our experimental methodology, including our experimental testbed setup and test scenario design. We then present the result analysis and discussion of each experiemtal scenario accordingly.

5.1. Experimental Testbed Setup

In our experimental testbed setup, as demonstrated in Figure 3, we configured the client, router, and server components to facilitate comprehensive experimentation. As shown in the Table 4, MikroTik RouterOS v6.49.4 serves as the network gateway, complemented by Ubuntu 22.04.4 deployed on both client machines. Network traffic generation is managed using Iperf3, while Wireshark captures data for detailed analysis. Our WiFi

hAP ax (see https://mikrotik.com/product/hap_ax3, accessed on 27 June 2024, for details) network configuration includes a 100 Mbps bottleneck and a queue buffer of 50 packets, utilizing the Pfifo queuing discipline. Clients connect via WiFi, ensuring flexibility and mobility, while the server maintains a stable Ethernet connection for robust data transmission. This setup allows us to rigorously test and analyze various algorithms such as CUBIC, BBR, PCC, BB2, and BBR3 over a duration of 100 s, focusing on metrics like throughput, throughout fairness Th_{ratio}, sending rate, packet loss ratio, RTT, and jitter.

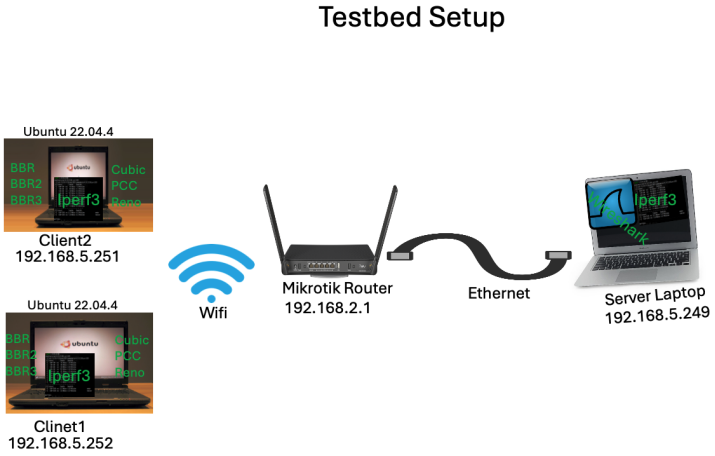


Figure 3. Our experimental testbed with 2 clients, one WiFi router, and one server where the clients are contending each other in the wireless bottleneck and the server is connected with ethernet.

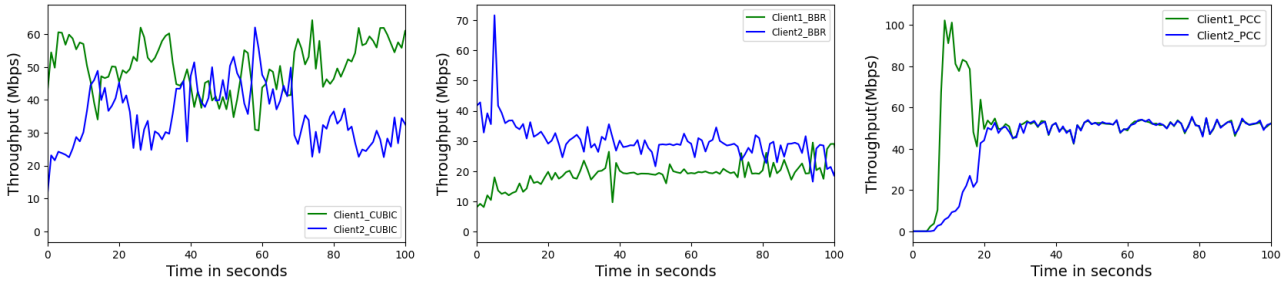
Table 4. Experimental setup and network configuration.

Testbed Setup	**Network Setup**
Operating System	Ubuntu 22.04.4
Traffic Generator	iperf3
Data Collection Tools	Wireshark, tshark, Scapy
RouterOS	MikroTik RouterOS 6.49.4; WiFi hAP ax (https://mikrotik.com/product/hap_ax3, accessed on 27 June 2024)
Network Configuration	
Bottleneck	100 Mbps
Queue Buffer	50 packets
Queue Discipline	PFIFO
TCP CCAs	CUBIC, BBR, PCC, BB2, BBR3
Experiment Duration	100 s
Metrics	Throughput, Sending Rate, Packet Loss Ratio, RTT, Jitter

We first observed the disparities in throughput within and between the CCAs (throughout fairness Th_{ratio}) and explored the RTT, jitter, and packet loss ratio within and between the different CCAs before and after switching in both line graphs and box plots. The aim was to minimize flow unfairness, optimize overall performance, and prevent flow starvation. Our investigation encompassed a range of CCAs, including CUBIC, BBR, Reno, PCC, BBR2, and BBR3. Each algorithm's behavior was thoroughly scrutinized across various experimental scenarios, evaluating the network stability, throughput, jitter, RTT, and packet loss ratio under different configurations. We conducted experiments involving specific intervals of CCA switching, such as transitioning from CUBIC, BBR, PCC, BBR2, and BBR3, to capture the intricate interactions between CCAs. Throughout our experiments, spanning 100 s with 1 s intervals, data collection and analysis using Wireshark and iperf3 statistics

enabled us to identify performance disparities and fairness issues among the tested algorithms. This rigorous analysis contributed insights toward optimizing and selecting algorithms tailored to diverse networking requirements.

The plots in Figure 4 highlight the behaviors of flows using the same CCA flows competing for bandwidth. Figure 4a, two CUBIC flows, and Figure 4b, two BBR flows, demonstrate that throughput disparities are prevalent within CCA flows, unlike in the case of PCC flows as shown in Figure 4c. Such throughput unfairness (quantifying throughout fairness using Th_{ratio}) observed in BBR and CUBIC flows suggests that both CCAs inherit complex and sensitive congestion mechanisms, which cannot always prevent unfairness among flows. Moreover, flow fairness varies in the real world as network conditions dynamically change, affecting each flow unevenly over time. PCC's focus on fairness through learning mechanisms explains its lower unfairness compared to other competitive CCAs.

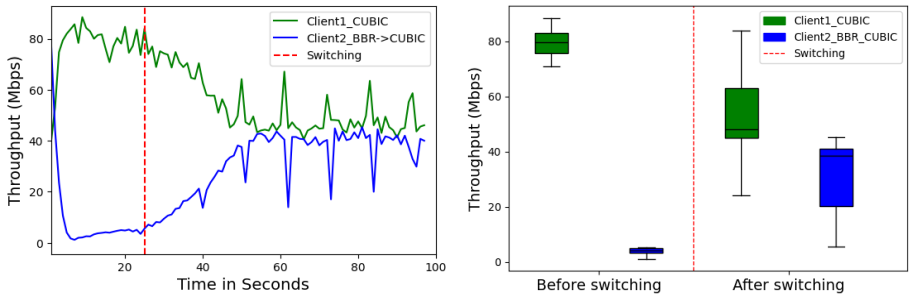


(**a**) CUBIC vs. CUBIC (**b**) BBR vs. BBR (**c**) PCC vs. PCC

Figure 4. Understanding throughput fairness (Th_{ratio}) by a comparison of throughput between two flows using the same CCA: (**a**–**c**). CUBIC flows showed higher fluctuations (**a**) compared to PCC flows (**b**), while BBR flows exhibited a significant throughput gap (**c**).

Results in Sections 5.2–5.4 show the complexities of fairness in multi-CCA network traffic. While all flows initially experience a slow throughput rise due to network learning (latency, jitter, bandwidth), different CCAs exhibit varying behaviors due to their inherent design philosophies. These design principles lead to unfairness when non-identical CCAs compete. Additionally, factors like differing responsiveness to network changes (e.g., sudden congestion events) and varying interpretations of "fairness" by different algorithms can further exacerbate unfairness. Understanding these dynamics is crucial for optimizing network performance. One potential avenue for improvement could be the development of "fairness-aware" CCAs or "switching" for fairer CCAs that can dynamically adjust their behavior based on the presence of other algorithms, promoting a more cooperative approach to resource allocation.

In Figure 5, CUBIC and BBR are competing, the throughput variation slightly improves over time. The influencing factor behind this is the adaptive nature of CCAs, which improves the estimation of available bandwidth. However, throughput imbalance remains unacceptably high in flows like in Figures 5. Not only do CCA flows such as CUBIC vs. BBR and PCC vs. BBR show disparities flattening out over time as CCAs adjust their data transfer aggressiveness, but distinct behaviors of CCAs with other CCA flows validate the hypothesis that no two CCA flows are perfectly fair to each other. Thus, intelligently assessing and switching available CCAs while studying changing network conditions would be very beneficial.



(**a**) (**b**)

Figure 5. Comparison of throughput between CUBIC and BBR, focusing on Client 2's switch to CUBIC at 25 s, understanding throughout fairness Th_{ratio}. Before the switch, Client 2 (BBR) had 4.08 Mbps compared to Client 1's 79.44 Mbps CUBIC flow. After switching, Client 2's average throughput increased to 38.72 Mbps, stabilizing close to Client 1's 40.64 Mbps CUBIC flow with minimal fluctuations. (**a**) CUBIC vs. BBR->CUBIC. (**b**) CUBIC vs. BBR->CUBIC.

Motivation to Choose the Scenarios

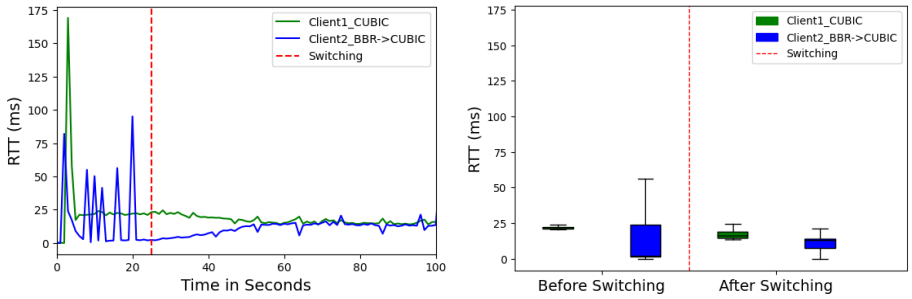
The selected scenarios CUBIC vs. BBR-CUBIC, PCC vs. BBR-PCC, and BBR vs. PCC-BBR were chosen to highlight the dynamic interplay between different CCAs under varying network conditions. These scenarios were selected due to their representative nature in demonstrating how dynamically switching between CCAs can optimize network performance, fairness, and resource utilization. CUBIC and BBR were chosen for their distinct approaches to congestion control, with CUBIC being widely used and BBR offering a newer, bandwidth-centric methodology. PCC was included for its adaptive nature and ability to handle diverse network conditions. By focusing on these specific pairings and transitions, the paper aims to illustrate practical benefits and challenges of dynamic CCA switching, providing insights into improving network performance and fairness in real-world scenarios.

5.2. Scenario 1: CUBIC vs. BBR-CUBIC

Dynamically switching CCAs based on network conditions significantly enhanced network performance and fairness among flows. The throughput graphs in Figure 5a clearly illustrate this improvement. After switching Client 2 from BBR to CUBIC at the 25 s mark, the throughput surged from near 0 Mbps (starvation) to approximately 40 Mbps by the 60 s mark and remained stable thereafter. This contrasts with the initial BBR behavior, which exhibited lower and more fluctuating throughput. Moreover, the box plot in Figure 5b confirms the significant increase in median throughput and reduced variability for Client 2 post-switching. This indicates a more equitable distribution of bandwidth among flows.

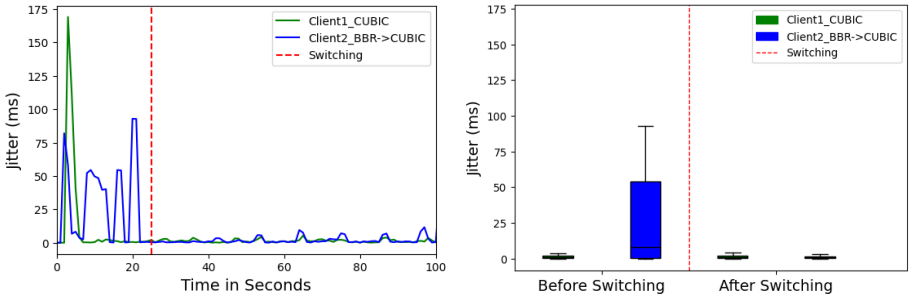
RTT is a critical factor influencing network fairness and congestion avoidance, which directly impacts throughput and sending rate. Despite a higher RTT, Client 1 achieved higher throughput at 30 s compared to 80 s in Figure 6a. This suggests that algorithms such as BBR and CUBIC optimize bandwidth utilization, mitigating the impact of high RTT on throughput. However, significant RTT fluctuations causes higher jitter variation as shown in Figure 7 degrade performance, as observed in the variability of throughput for Client 2 using BBR before switching in the Figures 5 and 6a,b.

The packet loss ratio reflects an algorithm's ability to manage congestion, showing a clear trend in Figure 8a. Before switching, BBR exhibited higher packet loss rates compared to CUBIC, despite sending fewer bytes. This indicates BBR's struggle to efficiently utilize available bandwidth under varying network conditions. After switching Figure 8b, Client 2 experienced reduced packet loss with CUBIC, while the competing flow saw a slight increase, suggesting CUBIC's better adaptability and congestion control capabilities.



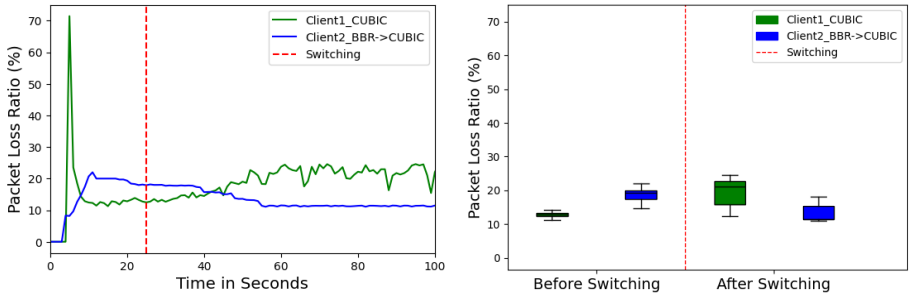
(**a**) CUBIC vs. BBR->CUBIC (**b**) CUBIC vs. BBR->CUBIC

Figure 6. RTT comparison of CUBIC vs. BBR->CUBIC, Client 2, before and after switching. RTT variation is significant: Client 1 (CUBIC) peaks at 170 ms at 5 s and drops to 20 ms, while Client 2 (BBR->CUBIC) ranges from 98 ms to 4 ms. Both reach 13 ms at 80 s (**a**). After switching, RTT variation and median values for both clients have decreased (**b**).



(**a**) CUBIC vs. BBR->CUBIC (**b**) CUBIC vs. BBR->CUBIC

Figure 7. Jitter comparison of Client 1 (CUBIC) and Client 2 (BBR->CUBIC) before and after switching at 25 s. Client 2's jitter varied widely, ranging from 90 ms to almost 0 ms. After switching (**a**), Client 2's average jitter dropped from 23 ms to 0.84 ms. Client 1's median jitter remained near 0 ms (**b**).



(**a**) CUBIC vs. BBR->CUBIC (**b**) CUBIC vs. BBR->CUBIC

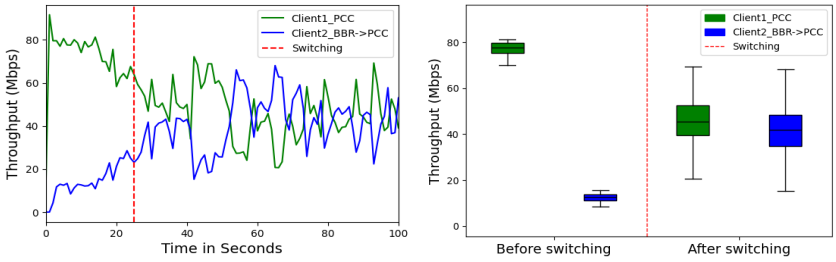
Figure 8. Packet loss ratio comparison of Client 1 (CUBIC) vs. Client 2 (BBR->CUBIC) before and after switching at 25 s. Client 2 (BBR) reduced its packet loss ratio from 19.35% to 11.45% after switching to CUBIC, while Client 1 (CUBIC) fluctuated from 70% to 11% at 5 and 8 s (**a**). The packet loss distribution for Client 2 (BBR), later CUBIC, had a median value of 20% before dropping to around 13%, with Client 1 experiencing a slight increase after switching (**b**).

Low jitter indicates stable network conditions without noticeable variation in packet arrival time. Inconsistent and higher jitter variations negatively impact the network, resulting in lower throughput. The relationship between jitter and throughput is illustrated in graphs (a) and (b) of Figures 5 and 6, respectively. It is noted that Client 2 experienced higher variations in RTT and packet arrival times before switching to CUBIC and received minimal throughput.

5.3. Scenario 2: PCC vs. BBR-PCC

Similar to CUBIC and BBR, PCC and BBR exhibited competitive behavior when contending for bandwidth under specific network conditions. Before the switch, Client 1 (PCC) generally outperformed Client 2 (BBR) in terms of throughput by observing throughout fairness Th_{ratio}. PCC's adaptive nature effectively adjusts to bandwidth and latency variations and contributed to its superior performance. Conversely, BBR demonstrated challenges in maintaining fairness, particularly in networks with varying RTTs.

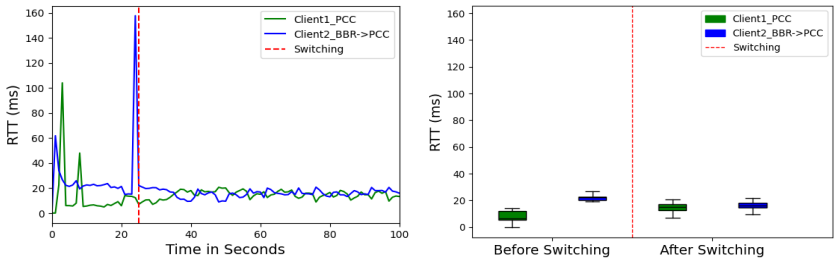
In Scenario 2, we investigated the impact of Client 2 switching from BBR to PCC at 25 s. Client 2 using BBR initially averaged throughput of 12.32 Mbps, but this surged to 40.40 Mbps after the switch to PCC in Figure 9a,b. This suggests that PCC offered a more efficient way to utilize bandwidth under these specific network conditions. BBR's congestion control strategy might not have been ideal, leading to underutilized resources.



(**a**) PCC vs. BBR->PCC (**b**) PCC vs. BBR->PCC

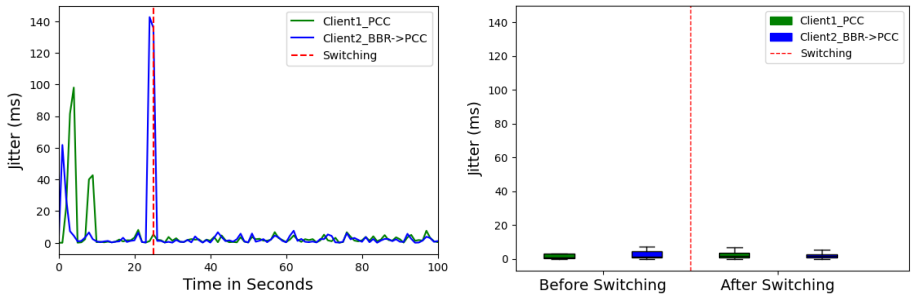
Figure 9. Comparison of the throughput of BBR and PCC, focusing on Client 2's switch to PCC at 25 s for understanding throughout fairness Th_{ratio}. Before switching, Client 2 (BBR) averaged 12.32 Mbps, which surged to 40.40 Mbps afterward (**a**). The distribution plot (**b**) shows Client 2's median throughput increasing from below 15 Mbps to above 35 Mbps.

Nevertheless, significant improvements were observed in latency and data loss. Client 2 (BBR) initially experienced high variations in RTT (Figure 10a). This variability suggests that BBR struggled to maintain consistent latency. Switching to PCC led to a decrease in these variations, with both clients maintaining RTTs below 20 ms. Similarly, jitter (variation in RTT) for Client 2 also decreased post-switch, as seen in Figure 11. These improvements suggest PCC's approach resulted in smoother data flow and lower latency for Client 2, possibly due to better congestion management.



(**a**) PCC vs. BBR->PCC (**b**) PCC vs. BBR->PCC

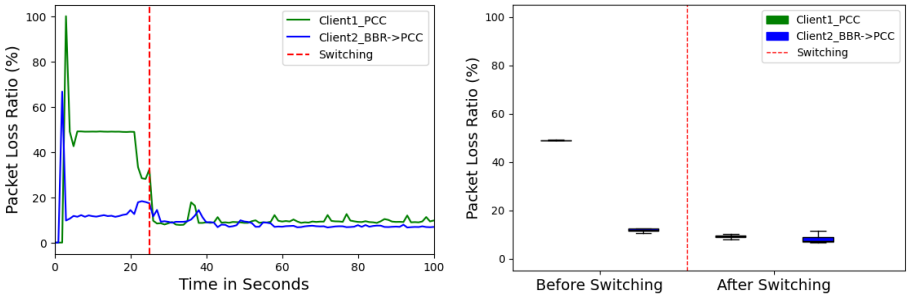
Figure 10. RTT variation over time: Client 1 (PCC) peaks at 102 ms at 3 s, dropping to 6 ms; Client 2 (BBR->PCC) ranges from 155 ms to 4 ms. Both clients maintained RTTs below 20 ms (**a**). Post-switch to Client 2 (PCC), RTT variation and median values decreased (**b**).



(**a**) PCC vs. BBR->PCC (**b**) PCC vs. BBR->PCC

Figure 11. Jitter comparison of Client 1 (PCC) and Client 2 (BBR->PCC) in milliseconds over time before and after switching at 25 s. Few fluctuations were observed for Client 1 (PCC) before switching (**a**). Median jitter remained near 1 ms for Client 1 after Client 2 switched to BBR->PCC (**b**).

Packet loss measurements also saw a dramatic improvement. Client 2 (BBR) displayed a high initial packet loss ratio (48.97%) (Figure 12a,b). This indicates significant data loss. Interestingly, Client 1 (PCC) also experienced a high initial packet loss. However, after the switch, the packet loss ratio for both clients dropped significantly. Client 2's median packet loss dropped to nearly 10%, while Client 1's (PCC) also improved to 7%. This suggests PCC's congestion control strategy effectively mitigated packet loss for both clients, likely by optimizing resource allocation and reducing network congestion.



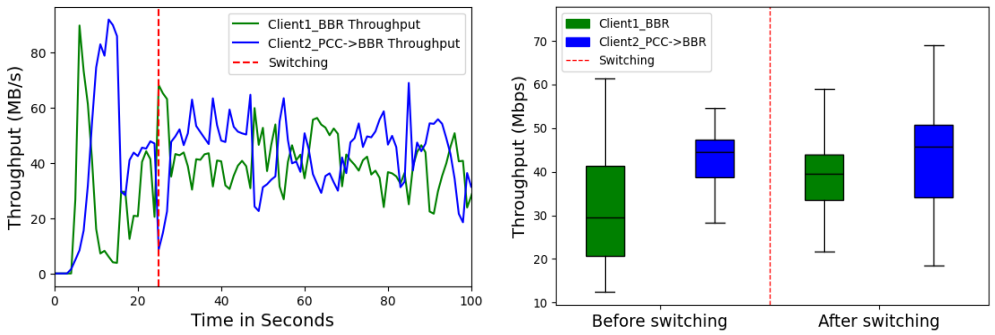
(**a**) PCC vs. BBR->PCC (**b**) PCC vs. BBR->PCC

Figure 12. Packet loss ratio comparison of Client 1 (PCC) vs. Client 2 (BBR->PCC) before and after switching at 25 s. The packet loss ratio for Client 1 (PCC) decreased from 48.97% to 9.22% after switching to Client 2 (PCC) in (**a**). The packet loss distribution for Client 2 (BBR), later PCC, had a median value of around 50% before dropping to nearly 10% (**b**).

5.4. Scenario 3: BBR vs. PCC-BBR

This analysis explores Scenario 3, where PCC and BBR compete for bandwidth. While PCC initially offered higher throughput for Client 2 (infer throughout fairness Th_{ratio}), significant fluctuations indicated potential instability. Switching Client 2 to BBR resulted in more stable performance for both clients across key metrics: throughput, RTT, jitter, and packet loss.

Initially, the competing CCA, PCC, provided Client 2 with significantly higher throughput, exceeding 44 Mbps on average, as seen in Figure 13a. However, this advantage came with substantial fluctuations in PCC's throughput, hinting at potential instability in its congestion control mechanism.

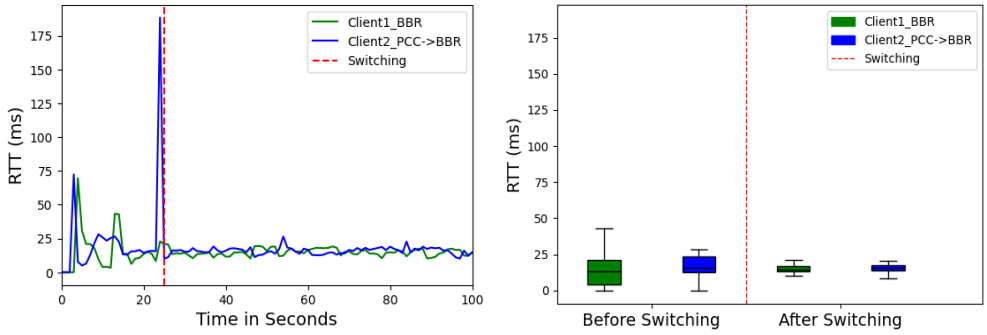


(**a**) BBR vs. PCC->BBR (**b**) BBR vs. PCC->BBR

Figure 13. Comparison of throughput and sending rate in Mbps over time before and after switching Client 2 (PCC->BBR) for an understanding of throughout fairness Th_{ratio}. Before switching to BBR at 25 s, Client 2 (PCC) had an average throughput of 44 Mbps, double that of the competing Client 1 (BBR) flow. After switching, Client 2 (BBR) received over 40 Mbps (**a**). The median value for Client 1 increased from 15 Mbps to 40 Mbps (**b**).

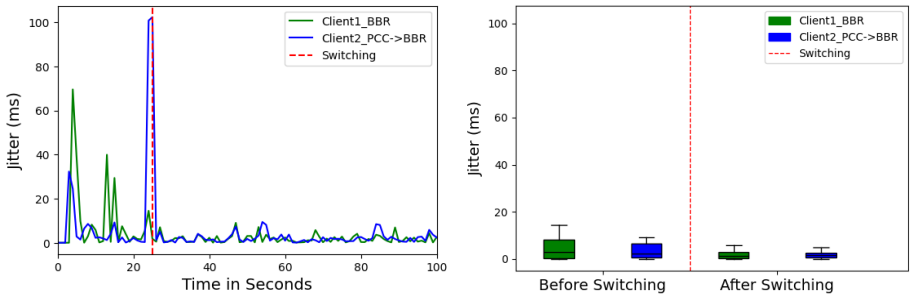
Client 2 switched from PCC to BBR, yielding positive impacts on network performance for both clients across several key metrics. Notably, Client 2 (BBR) maintained a stable throughput above 40 Mbps, demonstrating the effectiveness of BBR in achieving good throughput while ensuring stability. Interestingly, considering Figure 13b, even Client 1 (BBR) experienced improvement, with its median throughput rising to 40 Mbps. This suggests that the overall network congestion was better managed after the switch, potentially due to BBR's identical strategy to adapt network variation and optimize resource allocation.

Prior to the switch, Client 2 (PCC) exhibited significant spikes in RTT in Figure 14a, reaching as high as 104 ms at times. These fluctuations likely stemmed from the instability in PCC's throughput. After the switch, both clients displayed consistently lower and more stable RTTs under 15 ms. Similarly, jitter in Figure 15, which measures the variation in RTT, decreased significantly for both clients, with Client 2 (BBR) dropping to below 2 ms. This jitter reduction highlighted the benefit of stable throughput in maintaining consistent data flow and minimizing delays.



(**a**) BBR vs. PCC->BBR (**b**) BBR vs. PCC->BBR

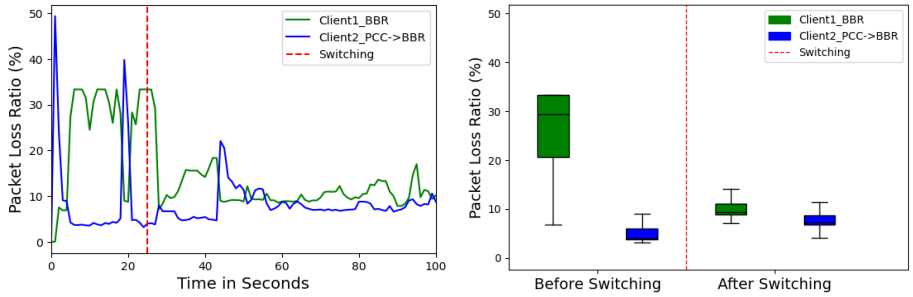
Figure 14. RTT variation over time: Client 2 (PCC) peaks at 104 ms at 3 s, dropping to 4 ms; Client 2 (PCC->BBR) ranges from 155 ms to 4 ms. Both clients maintained RTTs below 15 ms (**a**). Post-switch to Client 2 (BBR), RTT variation and median values decreased (**b**).



(**a**) BBR vs. PCC->BBR (**b**) BBR vs. PCC->BBR

Figure 15. Jitter comparison of Client 1 (BBR) and Client 2 (PCC->BBR) before and after switching at 25 s. Average jitter for Client 2 dropped from around 3 ms to nearly 1 ms after switching (**a**). Median jitter for Client 1 remained near 0 ms (**b**).

The observed improvements can be explained by considering the interrelationships between network performance metrics. High and unstable throughput in PCC could lead to buffer overflows, causing packet drops in Figure 16 and increased jitter in Figure 15. BBR's focus on congestion avoidance likely contributed to its lower initial throughput but resulted in greater stability across all metrics. This aligns with the established principle that stable throughput, even if slightly lower than the peak offered by an unstable CCA, leads to more efficient data flow and minimizes negative impacts on other metrics.



(**a**) BBR vs. PCC->BBR (**b**) BBR vs. PCC->BBR

Figure 16. Packet loss ratio comparison of Client 1 (BBR) vs. Client 2 (PCC->BBR) before and after switching at 25 s. Reduced packet loss ratio from 32% to 10% after the switch (**a**). The median packet loss for Client 1 (BBR) dropped from around 29% to near 10% (**b**).

Client 2 (PCC) also experienced a higher packet loss ratio compared to Client 1 (BBR) before the switch. This is likely because the unstable throughput of PCC could have resulted in buffer overflow and packet drops. Both clients benefited from reduced packet loss after the switch. Client 1 (BBR) displayed a more significant improvement, with median packet loss dropping from around 29% to near 10% in Figure 16. This further emphasizes the positive impact of BBR's congestion-avoidance strategy, which helps minimize packet drops by ensuring a smoother and more predictable data flow.

RTT and Jitter

Analyzing Scenarios 1, 2, and 3 reveals the critical role dynamic CCA selection plays in achieving fair resource allocation. Static CCAs can struggle to adapt, leading to underutilized bandwidth or flow starvation. A DRL model, when coupled real-time network data analysis, can choose the most suitable CCA for each flow, which prevents these issues.

Furthermore, fairness is crucial in ensuring a smooth user experience. A DRL model trained with fairness objectives can select CCAs that optimize individual flow performance while promoting fair resource distribution. This dynamic adaptation based on network conditions ensures optimal performance and user experience across all connections.

5.5. DRL Model

In the framework, we have employed a Deep Q-network (DQN) to optimize the switching of CCAs based on network performance metrics. The environment constructed using the OpenAI Gym framework has a state space comprising four key metrics: *throughput, latency, packet loss rate, and sending rate*. The action space was discrete, with only two possible actions: *switch the CCA or maintain the current CCA*.

The training data consisted of various network performance metrics recorded over time, representing different network states. Our custom environment, *CongestionControlEnv*, was designed to reset at the beginning of each episode, initializing the state with the current network conditions. At each step, the agent could either choose to switch the CCA based on an evaluation function or continue with the current CCA.

The evaluation function determines the best CCA by combining the current network state with historical data. It scores each available CCA based on weighted metrics of 70% for throughput, 20% for latency, and 10% for loss rate and selects the CCA with the highest score as the optimal choice for managing network congestion. This approach ensures that the most suitable algorithm is selected, effectively adapting to the dynamic nature of the network.

The agent was trained over 1000 episodes, with each episode allowing the agent to perform actions based on an epsilon-greedy policy, balancing exploration and exploitation. Experiences were stored in a replay buffer, from which the agent sampled mini-batches to train the Q-network. The Q-network, built using TensorFlow, consisted of two hidden layers with 64 neurons each, followed by an output layer corresponding to the action space. The training was performed using the mean squared error (MSE) loss function and the Adam optimizer. The reward function penalized high disparities in network performance metrics and the act of switching the CCA, encouraging the agent to maintain stable and optimal network conditions.

After training, the DQN model was evaluated using a separate test dataset to validate its performance. The test environment was configured similarly to the training environment, with network states derived from the test dataset. The trained model was loaded, and the agent's actions were recorded over 200 test episodes. The evaluation focused on the total reward, loss, and accuracy of the agent's actions.

In this paper, the DQN agent's neural network predicts actions by computing Q-values for each possible action given the current state. The action with the highest Q-value is selected as the predicted action. These predicted actions are then executed by the agent in the environment, which responds by transitioning to a new state and providing a reward based on the action taken. This dynamic interaction between the agent and the environment allows the DQN to tackle the complex task of selecting the most suitable CCA in ever-changing network environments.

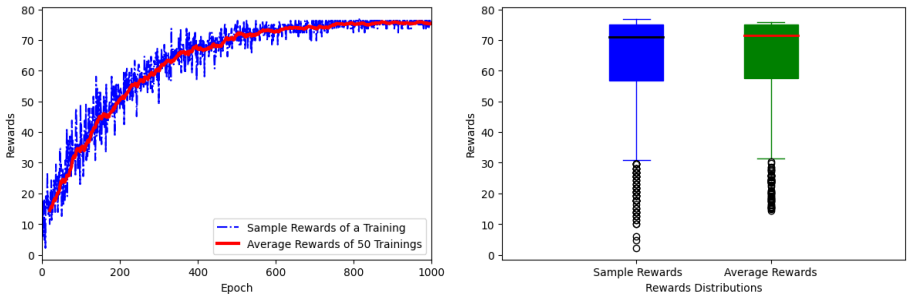
To evaluate the performance of our DQN, we primarily focused on cumulative reward as a metric. However, we also incorporated accuracy as a supplementary metric, following the approach of [56]. Accuracy is calculated by comparing the predicted actions of the DQN agent to the actual actions taken by the environment, providing insight into how closely the agent's decisions align with a reference policy during training.

Additionally, loss is computed as the mean squared error between the predicted Q-values and the target Q-values. These target Q-values are derived from the reward received and the maximum predicted Q-values of the next state, ensuring the DQN learns effectively over time. This approach not only monitors the agent's learning process but also provides a more comprehensive evaluation of the DQN's effectiveness, as emphasized by [57], in real-time TCP congestion control scenarios.

5.5.1. Training Reward

Figure 17a, showing the convergence of training rewards, illustrates the evolution of training rewards over 1000 epochs during the training of the DRL agent. The y-axis represents the convergence rewards accumulated by the agent in each epoch, while the

x-axis corresponds to the number of training epochs. The blue curve shows the sample rewards from a single training run, reflecting the variability in the agent's learning process, while the red curve represents the average rewards over 50 training runs, providing a smoothed view of the agent's performance over time.



(**a**) Convergence of training rewards (**b**) Distribution of training rewards

Figure 17. Visualization of training a DRL model: The total rewards over 1000 epochs demonstrate consistent improvement, with performance approaching a near-optimal policy after 700 epochs, where the rewards stabilize around 75, as shown in line graph (**a**). The box plot (**b**) reveals that the majority of training epochs achieve an average reward of 70, which represents the median value.

The curves indicate a general upward trend, signifying the agent's progress towards a near-optimal policy. As the epochs increase, the total rewards rise, demonstrating that the agent is improving its decision-making capabilities. However, the blue curve exhibits fluctuations and occasional plateaus, suggesting that the learning process involves an exploration of the state–action space, which is inherent to DRL algorithms. The eventual stabilization of the red curve toward the latter epochs suggests that the agent is converging towards an optimal policy, where further improvements become marginal.

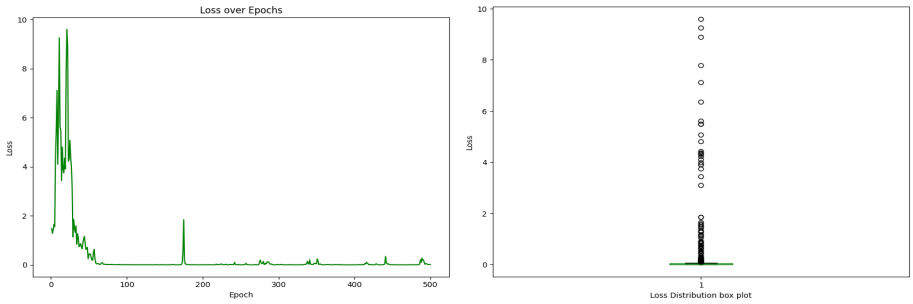
Figure 17b, showing the distribution of training rewards, visualizes the distribution of rewards across different training episodes. The blue boxplot represents the sample rewards from a single training run, while the green boxplot illustrates the average rewards across 50 training runs.

The median reward, located around 70, signifies the typical performance level of the agent during training. The small inter-quartile range (IQR) suggests that the agent's performance was consistent across episodes, with the rewards generally clustering around the median. There are a few outliers visible at the lower end of the range, indicating instances where the agent received significantly lower rewards. However, the presence of these outliers is limited, suggesting that the agent was mostly successful in selecting high-performing CCAs that matched the current network conditions. This consistency in performance highlights the model's effectiveness in minimizing network issues like unfairness and flow starvation.

5.5.2. Training Loss

The downward trajectory of the loss curve in Figure 18a indicates successful learning by the DRL agent, progressively minimizing prediction errors over the training epochs. Specifically, after 58 epochs, the loss becomes minimal, hovering close to zero with slight fluctuations observed between epochs 176 to 179. This pattern suggests that the trained model effectively identified and switched between the most suitable congestion control algorithms (CCAs) for current network conditions, thereby maintaining flow fairness and preventing starvation.

The corresponding box plot in Figure 18b illustrates fluctuations in loss values, indicating challenges encountered during training. However, the fact that 88% of the loss distribution is negligible (close to zero) underscores the agent's ability to develop a robust policy.

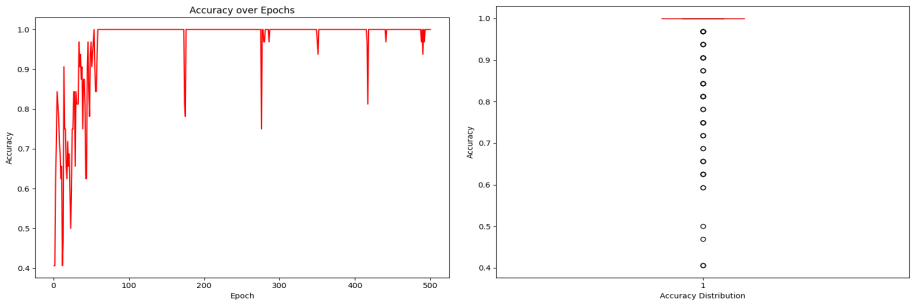


(**a**) Training loss curve (**b**) Training loss distribution

Figure 18. Visualization of training a DRL model: The loss over 500 epochs, shown in line graph (**a**), approaches near zero after 58 epochs, with minimal fluctuations between epochs 176 and 179. The box plot (**b**) indicates that 88% of the loss distribution is negligible (close to zero).

5.5.3. Training Accuracy

The accuracy graph in Figure 19a shows an exponential upward trend, indicating highly accelerated improved decision-making by the DRL agent over time, with the model reaching 100% accuracy after the 60-epoch mark. As depicted in Figure 19b, minor fluctuations and outliers were typical during training and did not significantly detract from the overall upward trajectory. The fact that 96.8% of epochs achieved 100% accuracy highlights the model's capability to make precise decisions and switch to the best available CCAs. This trend suggests effective learning and policy improvement by the agent. However, the fluctuations in accuracy during training indicate that there is still room for improvement, which represents a limitation of this model.



(**a**) Evolution of training accuracy (**b**) The distribution of the training accuracy

Figure 19. Visualization of training a DRL model: The accuracy over 500 epochs, as shown in line graph (**a**), attains 100% after the 60-epoch mark. The box plot (**b**) shows the distribution of training accuracy, with 96.8% of epochs achieving 100% accuracy.

5.6. Our Key Findings

Table 5 summarizes our key findings with the performance metrics of various CCAs over WiFi, comparing scenarios before and after switching the congestion control.

In Scenario 1, Client 1 (CUBIC) demonstrates superior throughput (79.44 Mbps) compared to Cleint 2 (BBR) before switching (4.08 Mbps), but at the cost of higher RTT (21.44 ms vs. 2.32 ms) and jitter (0.75 ms vs. 23.99 ms), with a packet loss of 12.8% versus BBR's 19.35%. After switching, Client 1's throughput decreased to 40.64 Mbps, while Client 2's improved to 38.72 Mbps, with RTTs of 16.30 ms and 13.01 ms, respectively. In Scenario 2, Cleint 1 (PCC) exhibited higher throughput (77.36 Mbps) before switching compared to Client 2 (BBR) (12.32 Mbps) but sufferd from greater packet loss (48.97% vs. 11.83%). After switching, Client 1's throughput decreased to 44 Mbps, while its packet loss improved to 9.22%, compared to Client 2's throughput of 40.4 Mbps and loss of 7.21%. In Scenario 3, Client 1 (BBR) before switching shows a throughput of 20.48 Mbps and RTT of 13.40 ms,

with a packet loss of 29.44%. Post-switching, BBR's performance dramatically improved to 42.56 Mbps in throughput, 15.78 ms in RTT, and 7.20% in packet loss.

Table 5. Summary of results from network emulation experiments.

Our Setup	CCAs over WiFi Network Settings	2 Clients (C1, C2)	Aggregate Performance Metrics			
			Throughput	RTT	Jitter	Loss
Scen. 1	CUBIC vs. BBR (before)	CUBIC(C1)	79.44 Mbps	21.44 ms	0.75 ms	12.8%
		BBR(C2)	4.08 Mbps	2.32 ms	23.99 ms	19.35%
	CUBIC vs. BBR->CUBIC (after)	CUBIC(C1)	40.64 Mbps	16.30 ms	1.09 ms	21.10%
		CUBIC(C2)	38.72 Mbps	13.01 ms	0.82 ms	11.45%
Scen. 2	PCC vs. BBR (before)	PCC(C1)	77.36 Mbps	6.27 ms	0.84 ms	48.97%
		BBR(C2)	12.32 Mbps	22.02 ms	1.163 ms	11.83%
	PCC vs. BBR->PCC (after)	PCC(C1)	44 Mbps	15.08 ms	1.722 ms	9.22%
		PCC(C2)	40.4 Mbps	15.98 ms	1.53 ms	7.21%
Scen. 3	BBR vs. PCC (before)	BBR (C1)	20.48 Mbps	13.40 ms	2.85 ms	29.44%
		PCC(C2)	42.08 Mbps	15.76 ms	2.14 ms	4.17 %
	BBR vs. PCC->BBR (after)	BBR(C1)	42.56 Mbps	14.63 ms	1.16 ms	9.35%
		BBR (C2)	45.6 Mbps	15.78 ms	1.53 ms	7.20%

Table 5 highlights the varied impacts of different CCAs and switching on throughput, RTT, jitter, and packet loss, demonstrating the need for dynamic adaptation to optimize network performance. Upon reviewing several rigorous CCA studies in the field, including their driving principles, behaviors, and performance in different network conditions summarized in Table 5, we draw the following findings:

- *Mitigating Non-Congestive Delay Variations:* Delay-bounding CCAs aim to manage non-congestive delay variations by ensuring that delay adjustments comprise at least half the expected non-congestive jitter along the network path [18]. If the delay oscillations fall below this threshold, the CCA may struggle to maintain high throughput, bounded delays, and fairness, potentially leading to inefficient network performance.
- *Characteristics and Thresholds for Network Design:* CCAs should adjust delays by at least half the expected non-congestive delays to differentiate between congestion-related and other delays. Failing to meet this threshold can cause the CCA to struggle with throughput, delay management, and fairness [18].
- *Dynamic Switching of CCAs and Insights into Fairness and Stability:* This study reveals that throughput unfairness persists within the same CCA, influenced by network path characteristics. Dynamically switching between CCAs based on network conditions can improve fairness and stability. These findings highlight the potential of using Deep Reinforcement Learning to adapt CCAs dynamically for better network performance and fairness.

6. Conclusions and Future Work

This paper presents an approach to improving the fairness of TCP CCA in WiFi networks using DRL. To address unfairness and potential starvation issues in traditional CCAs, we developed a dynamic CCA switching mechanism that uses offline DRL to select the most appropriate CCA based on real-time network conditions. Our analysis and experiments indicate that this approach can help reduce congestion and improve fairness among competing network flows. Key aspects of our work include delay variation analysis, the development of a CCA switching mechanism, and validation in a testbed environment. The results show improvements in throughput stability, reduced RTT and jitter, and lower packet loss, contributing to better overall network performance. Considering both QoS and QoE metrics, our approach aims to balance technical performance with user experience.

The study highlights the potential of DRL for adaptive network management, with offline DRL offering a practical implementation path without the complexities of online learning.

Our work has certain limitations. The model was tested in offline environments with precollected WiFi data, and its real-time performance in hybrid or wired networks has not been explored. Furthermore, the study focused on common TCP variants such as Cubic, PCC, BBR, and Copa, which may not extend to machine-learning-based TCP algorithms such as Remy, Aurora, and Astraea. Future research should address these limitations by testing the model in real-time network environments, integrating active queue management, developing CCAs for noncongestive delay jitter, and using data from a broader range of machine learning-based TCP algorithms to improve generalization.

Author Contributions: Conceptualization, J.K.; Methodology, S.K.S., S.R.P. and J.K.; Software, S.K.S., S.R.P. and J.K.; Formal analysis, S.R.P. and J.K.; Investigation, S.K.S., S.R.P. and J.K.; Writing—review & editing, S.K.S. and S.R.P.; Visualization, S.K.S.; Supervision, S.R.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Detailed Calculations

Appendix A.1. TCP CUBIC

As noted earlier in Section 3, TCP CUBIC dynamics can be mathematically modeled by using the congestion window (W) as follows. W is time-varying; therefore, $W(t)$ evolves as

$$W(t) = \left(s(t) - \sqrt[3]{\frac{W_{\max}(t)\beta}{C}}\right)^3 + W_{\max}(t) \tag{A1}$$

where $s(t)$ is the elapsed time since the last window reduction, $W(t)$ is the current congestion window at time t, and C refers to scaling constant and β is multiplicative decrease factor.

Using Equation (A1) with the fixed-point approach along the lines of [58], we obtain the equilibrium congestion window, $W^\star$, as

$$W^\star = \sqrt[4]{\frac{\tau^3 C}{(p^\star)^3 \beta}} \tag{A2}$$

We consider to cases to explain the causes of unfairness and starvation.

CASE I: Considering two CUBIC flows with different RTTs, τ_1 and τ_2, and experiencing the same loss, p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$W_1^\star = \sqrt[4]{\frac{\tau_1^3 C}{(p^\star)^3 \beta}};\ W_2^\star = \sqrt[4]{\frac{\tau_2^3 C}{(p^\star)^3 \beta}}$$

$$\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{\tau_1^3}{\tau_2^3}} \tag{A3}$$

Using (A3), if $\tau_1 = 2\tau_2$, then $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{8}$, which explains the key reason for the observed unfairness.

CASE II: Considering two CUBIC flows with the same RTTs, τ, and experiencing different losses, p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve:

$$W_1^\star = \sqrt[4]{\frac{\tau^3 C}{(p_1^\star)^3 \beta}};\ W_2^\star = \sqrt[4]{\frac{\tau^3 C}{(p_2^\star)^3 \beta}}$$

$$\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{p_2^3}{p_1^3}} \tag{A4}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{1}{8}}$.

As we know, **Starvation** is an extreme case of unfairness,and the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = 1, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{\frac{1}{n^3}}$;

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \sqrt[4]{n^3}$.

Appendix A.2. PCC Vivace

PCC Vivace separates time into consecutive monitor intervals (MIs) [14,15,37]. Each MI is responsible for transforming performance statistics presented at that MI to a numerical utility value.

$$U = s_i - (bs_i)\frac{d(\tau_i)}{dT} - cs_i L_i \tag{A5}$$

where $\frac{dRTT_i}{dT} = \frac{d\tau_i}{dT}$ is the RTT gradient in an MI, s_i is sender i's the sending rate, L_i is the loss rate, T is the time unit, and b and c are constants $(b \geq 0, c < 1)$.

Using Equation (A5) with a fixed-point approach along the lines of [18], we obtain the equilibrium $U^\star$ (utilityfunction) as

$$U^\star = \frac{W_1}{\tau_i} - b\frac{W_1}{\tau_i}\frac{d\tau_i}{dT} - c\frac{W_1}{\tau_i}L_i$$

$$U^* = 1 - b\frac{d\tau_i}{dT} - cL_i \tag{A6}$$

We consider two cases to explain the causes of unfairness and starvation.

CASE I: Considering two PCC flows with different RTTs, τ_1 and τ_2, and experiencing the same loss, p, we can evaluate how the congestion windows $U_1^\star$ and $U_2^\star$ evolve as

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - b\frac{d\tau_1}{dT} - cL_i}{1 - b\frac{d\tau_2}{dT} - cL_i} \tag{A7}$$

If $\tau_1 = 2\tau_2$, then $\frac{U_1^\star}{U_2^\star} = \frac{1-2b\frac{d\tau_2}{dT} - cL_i}{1-b\frac{d\tau_2}{dT} - cL_i}$

CASE II: Considering two PCC flows with same RTTs, τ, and experiencing different losses p_1 and p_2, we can evaluate how the congestion windows $U_1^\star$ and $U_2^\star$ evolve:

$$U_1^\star = 1 - b\frac{d\tau}{dT} - cL_i$$

$$U_2^\star = 1 - b\frac{d\tau}{dT} - cL_i$$

$$\frac{U_1^\star}{U_2^\star} = \frac{(1 - b\frac{d\tau}{dT} - cL_i)(1 - p_2)}{(1 - b\frac{d\tau}{dT} - cL_i)(1 - p_1)} = \frac{1 - p_2}{1 - p_1} \tag{A8}$$

If $p_1 = 2p_2$, then $\frac{U_1^\star}{U_2^\star} = \frac{1-p_2}{1-2p_2}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = \{1, 2, 3, \ldots, 20\}$

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - p_2}{1 - np_2}$$

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, \ldots, 20$

$$\frac{U_1^\star}{U_2^\star} = \frac{1 - bn\frac{d\tau_2}{dT} - cL_i}{1 - b\frac{d\tau_2}{dT} - cL_i}$$

Appendix A.3. BBR

During this period, the delivery rate remains unchanged but RTT increases. Moreover, BBR limits the data pacing rate, also known as the sending rate, and inflight data to one BDP to control the congestion [47,48], where

$$BDP = Btlbw.RTprop$$

where RTT_{prop} refers to Round Trip Propagation Time.

BBR considers the most recent estimated delivery rate and RTT to obtain its transmission capability. In BBR, the current Btlbw is the maximum delivery rate of the last 10 RTT, and the current RTT_{prop} is the maximum delay calculated in the past 10 s [47,48]. BBR adjusts the pacing and CWND rate as follows, as mentioned in [47]:

$$pacing\_rate = pacing\_gain.Btlbw \tag{A9}$$

$$CWND = cwnd\_gain.BDP \tag{A10}$$

Using a fixed-point approach along the lines of [18], we obtain the equilibrium congestion window $W^\star$ as

$$W^\star = \frac{\alpha\tau^\star}{\tau^\star - 2\tau_m} \tag{A11}$$

τ_m is the minimum RTT.

We consider cases to explain the causes of unfairness and starvation.

CASE I: Considering two BBR flows with different RTTs, τ_1 and τ_2, and experiencing the same loss, p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$W_1^\star = \frac{\alpha\tau_1}{\tau_1 - 2\tau_m};\ W_2^\star = \frac{\alpha\tau_2}{\tau_2 - 2\tau_m}$$

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau_1}{\tau_2}\left(\frac{\tau_2 - 2\tau_m}{\tau_1 - 2\tau_m}\right) \tag{A12}$$

If $\tau_1 = 2\tau_2$, then

$$\frac{W_1^\star}{W_2^\star} = \frac{2\tau_2}{\tau_2}\left(\frac{\tau_2 - 2\tau_m}{2\tau_2 - 2\tau_m}\right)$$

CASE II: Considering BBR flows with the same RTTs, τ, and experiencing different losses, p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve:

$$W_1^\star = \frac{\alpha\tau_1}{\tau_1 - 2\tau_m};\ W_2^\star = \frac{\alpha\tau_2}{\tau_2 - 2\tau_m}$$

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau - 2\tau_m(1-p_2)}{\tau - 2\tau_m(1-p_1)} \tag{A13}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-2p_2}$.

As we know, **Starvation** is an extreme case of unfairness; the two cases discussed above will have the following consequences:

CASE I: If $p_1 = np_2$, where $n = 1, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-np_2}$

CASE II: If $\tau_1 = n\tau_2$, where $n = 1, \ldots, 20$ $\frac{W_1^\star}{W_2^\star} = \frac{3n}{5n-2}$

Appendix A.4. COPA

As mentioned, the first task is to set a target rate and achieve it, which can be expressed as

$$\lambda = \frac{1}{\delta d_q} \tag{A14}$$

where λ is the sending rate, d_q is the measured queuing delay in seconds, and $\frac{1}{\delta}$ is the unit of packet size. The second is the window update rule that encourages the sender to move toward the target rate. Finally, the TCP competitive strategy is to rival buffer-filling flows.

$$\lambda = \frac{W}{\tau} \tag{A15}$$

where τ is the smallest observed RTT in the current window, where $\tau = srtt/2$ [16,59]. τ is considered as RTTstanding at [16]. τ also refers to the RTT corresponding to a standing queue as it is the smallest observed RTT in the recent window, and srtt refers to the value of the standard smoothed RTT estimation.

Copa demands the sender to track the target rate, and they do so by using Equation (A14). The sender also estimates the queuing delay with the following equation:

$$d_q = \tau - T_m \tag{A16}$$

where T_m is the minimum RTT measured over a long period.

Using Equations (A14)–(A16) with a fixed-point approach along the lines of [18], we obtain the equilibrium congestion window $W^\star$ as

$$W^\star = \frac{\tau^\star}{\delta(\tau^\star - T_m)} \tag{A17}$$

We consider cases to explain the causes of unfairness and starvation.

CASE I: Considering two Copa flows with different RTTs, τ_1 and τ_2, and experiencing the same loss, p, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve as

$$W_1^\star = \frac{\tau_1}{\delta(\tau_1 - T_m)};\ W_2^\star = \frac{\tau_2}{\delta(\tau_2 - T_m)}$$

$$\frac{W_1^\star}{W_2^\star} = \frac{\tau_1}{\tau_2}\left(\frac{\tau_2 - T_m}{\tau_1 - T_m}\right) \tag{A18}$$

If $\tau_1 = 2\tau_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{2\tau_2}{\tau_2}\left(\frac{\tau_2 - T_m}{2\tau_2 - T_m}\right)$.

CASE II: Considering Copa flows with same RTTs, τ, and experiencing different losses, p_1 and p_2, we can evaluate how the congestion windows $W_1^\star$ and $W_2^\star$ evolve:

$$W_1^\star = \frac{\tau_1}{\delta(\tau_1 - T_m)};\ W_2^\star = \frac{\tau_2}{\delta(\tau_2 - T_m)}$$

$$\frac{W_1^\star}{W_2^\star} = \frac{(\tau - T_m)(1-p_2)}{(\tau - T_m)(1-p_1)} = \frac{1-p_2}{1-p_1} \tag{A19}$$

If $p_1 = 2p_2$, then $\frac{W_1^\star}{W_2^\star} = \frac{1-p_2}{1-2p_2}$.

References

1. 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]
2. Kua, J.; Armitage, G.; Branch, P. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1842–1866. [CrossRef]
3. Zhang, J.; Zhang, Y.; Dong, E.; Zhang, Y.; Ren, S.; Meng, Z.; Xu, M.; Li, X.; Hou, Z.; Yang, Z.; et al. Bridging the Gap between QoE and QoS in Congestion Control: A Large-scale Mobile Web Service Perspective. In Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 23), Boston, MA, USA, 10–12 July 2023; pp. 553–569.
4. Hoe, J.C. Improving the start-up behavior of a congestion control scheme for TCP. *ACM SIGCOMM Comput. Commun. Rev.* **1996**, *26*, 270–280. [CrossRef]
5. 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]
6. Tan, K.; Song, J.; Zhang, Q.; Sridharan, M. A compound TCP approach for high-speed and long distance networks. In Proceedings of the IEEE INFOCOM, Barcelona, Spain, 23–29 April 2006.
7. Pokhrel, S.R.; Williamson, C. Modeling compound TCP over WiFi for IoT. *IEEE/ACM Trans. Netw.* **2018**, *26*, 864–878. [CrossRef]
8. Floyd, S. TCP and explicit congestion notification. *ACM SIGCOMM Comput. Commun. Rev.* **1994**, *24*, 8–23. [CrossRef]
9. Winstein, K.; Sivaraman, A.; Balakrishnan, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), Berkeley, CA, USA, 2–5 April 2013; pp. 459–471.
10. Winstein, K.; Balakrishnan, H. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 123–134. [CrossRef]
11. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of the Conference on Communications Architectures, Protocols and Applications, London, UK, 31 August–2 September 1994; pp. 24–35.
12. Wei, D.X.; Jin, C.; Low, S.H.; Hegde, S. FAST TCP: Motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.* **2006**, *14*, 1246–1259. [CrossRef]
13. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *Commun. ACM* **2017**, *60*, 58–66. [CrossRef]
14. Dong, M.; Meng, T.; Zarchy, D.; Arslan, E.; Gilad, Y.; Godfrey, B.; Schapira, M. PCC vivace:Online-Learning congestion control. In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), San Francisco, CA, USA, 16–18 April 2018; pp. 343–356.
15. Jay, N.; Gilad, T.; Frankel, N.; Meng, T.; Godfrey, B.; Schapira, M.; Chung, J.W.; Siwach, V.; Salim, J.H. A PCC-Vivace Kernel Module for Congestion Control. University of Illinois Urbana-Champaign, Hebrew University of Jerusalem in Israel, Verizon. 2018. Available online: https://pbg.web.engr.illinois.edu/papers/jay18pcc-kernel.pdf (accessed on 2 September 2024)
16. 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; pp. 329–342.
17. Zaki, Y.; Pötsch, T.; Chen, J.; Subramanian, L.; Görg, C. Adaptive congestion control for unpredictable cellular networks. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, UK, 17–21 August 2015; pp. 509–522.
18. Arun, V.; Alizadeh, M.; Balakrishnan, H. Starvation in end-to-end congestion control. In Proceedings of the ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, 22–26 August 2022; pp. 177–192.
19. Seo, S.J.; Cho, Y.Z. Fairness enhancement of TCP congestion control using reinforcement learning. In Proceedings of the 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Republic of Korea, 21–24 February 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 288–291.
20. Liao, X.; Tian, H.; Zeng, C.; Wan, X.; Chen, K. Towards fair and efficient learning-based congestion control. *arXiv* **2024**, arXiv:2403.01798.
21. Pokhrel, S.R.; Panda, M.; Vu, H.L. Fair Coexistence of Regular and Multipath TCP over Wireless Last-Miles. *IEEE Trans. Mob. Comput.* **2019**, *18*, 574–587. [CrossRef]
22. Hamzah, M.F.; Athab, O.A. A Review of TCP Congestion Control Using Artificial Intelligence in 4G and 5G Networks. *Am. Acad. Sci. Res. J. Eng. Technol. Sci.* **2022**, *88*, 172–186.
23. Pokhrel, S.R.; Panda, M.; Vu, H.L.; Mandjes, M. TCP Performance over Wi-Fi: Joint Impact of Buffer and Channel Losses. *IEEE Trans. Mob. Comput.* **2016**, *15*, 1279–1291. [CrossRef]
24. Wang, L. Low-Latency, High-Throughput Load Balancing Algorithms. *J. Comput. Technol. Appl. Math.* **2024**, *1*, 1–9.

25. Haile, H.; Grinnemo, K.J.; Ferlin, S.; Hurtig, P.; Brunstrom, A. End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks. *Comput. Netw.* **2021**, *186*, 107692. [CrossRef]
26. Kua, J.; Nguyen, S.H.; Armitage, G.; Branch, P. Using active queue management to assist IoT application flows in home broadband networks. *IEEE Internet Things J.* **2017**, *4*, 1399–1407. [CrossRef]
27. Pokhrel, S.R.; Kua, J.; Satish, D.; Ozer, S.; Howe, J.; Walid, A. DDPG-MPCC: An Experience Driven Multipath Performance Oriented Congestion Control. *Future Internet* **2024**, *16*, 37. [CrossRef]
28. Satish, D.; Kua, J.; Pokhrel, S.R. Active Queue Management in L4S with Asynchronous Advantage Actor-Critic: A FreeBSD Networking Stack Perspective. *Future Internet* **2024**, *16*, 265. [CrossRef]
29. Liu, Q.; Yang, P.; Yang, M.; Yu, L. CKCD: A fair and low latency queue control algorithm for heterogeneous TCP flows. In Proceedings of the 2020 International Conference on Computing, Networking and Communications (ICNC), Big Island, HI, USA, 17–20 February 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 725–730.
30. Bazi, K.; Nassereddine, B. Comparative analysis of TCP congestion control mechanisms. In Proceedings of the 3rd International Conference on Networking, Information Systems & Security, Marrakech, Morocco, 31 March–2 April 2020; pp. 1–4.
31. Gettys, J. Bufferbloat: Dark buffers in the Internet. *IEEE Internet Comput.* **2011**, *15*, 96. [CrossRef]
32. Ye, J.; Leung, K.C. Adaptive and stable delay control for combating bufferbloat: Theory and algorithms. *IEEE Syst. J.* **2019**, *14*, 1285–1296. [CrossRef]
33. McNair, D.S. Preventing disparities: Bayesian and frequentist methods for assessing fairness in machine learning decision-support models. In *New Insights Into Bayesian Inference*; IntechOpen: London, UK, 2018; Volume 71.
34. Kang, M.; Li, L.; Weber, M.; Liu, Y.; Zhang, C.; Li, B. Certifying some distributional fairness with subpopulation decomposition. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 31045–31058.
35. Valli, S.; Sankar, S.; Mehata, K. A Heuristic Method for Improving Tcp Performance by a Greedy Routing Algorithm. *J. Theor. Appl. Inf. Technol.* **2017**, *95*, 5215–5223.
36. Yamazaki, M.; Yamamoto, M. Fairness improvement of congestion control with reinforcement learning. *J. Inf. Process.* **2021**, *29*, 592–595. [CrossRef]
37. Zhang, S.; Lei, W.; Zhang, W.; Li, H. An evaluation of bottleneck bandwidth and round trip time and its variants. *Int. J. Commun. Syst.* **2021**, *34*, e4772. [CrossRef]
38. Xiao, K.; Mao, S.; Tugnait, J.K. TCP-Drinc: Smart congestion control based on Deep Reinforcement Learning. *IEEE Access* **2019**, *7*, 11892–11904. [CrossRef]
39. Ke, C.H.; Astuti, L. Applying Deep Reinforcement Learning to improve throughput and reduce collision rate in IEEE 802.11 networks. *KSII Trans. Internet Inf. Syst. (TIIS)* **2022**, *16*, 334–349.
40. Kim, M.; Hwang, S.; Lee, I. Deep Reinforcement Learning approach for fairness-aware scheduling in wireless networks. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1229–1232.
41. Arianpoo, N.; Leung, V.C. How network monitoring and reinforcement learning can improve tcp fairness in wireless multi-hop networks. *EURASIP J. Wirel. Commun. Netw.* **2016**, *2016*, 278. [CrossRef]
42. Yu, Y.; Wang, T.; Liew, S.C. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1277–1290. [CrossRef]
43. Maeta, K.; Kitagata, G.; Hasegawa, G. Improving per-flow fairness by ML-based estimation of competing flows' congestion control algorithm. In Proceedings of the 2022 13th International Conference on Ubiquitous and Future Networks (ICUFN), Barcelona, Spain, 5–8 July 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 376–381.
44. Jay, N.; Rotman, N.; Godfrey, B.; Schapira, M.; Tamar, A. A Deep Reinforcement Learning perspective on Internet congestion control. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; PMLR: Birmingham, UK, 2019; pp. 3050–3059.
45. Naqvi, H.A.; Hilman, M.H.; Anggorojati, B. Implementability improvement of Deep Reinforcement Learning based congestion control in cellular network. *Comput. Netw.* **2023**, *233*, 109874. [CrossRef]
46. Giacomoni, L. Enhancing Design and Evaluation Methods for Reinforcement Learning-based Congestion Control: A Large Scale Experimental Study of Fairness, Efficiency, Responsiveness and a Novel Simulation Framework as a Training and Evaluation Playground. Ph.D. Thesis, University of Sussex, Sussex, UK, 2024. Available online: https://hdl.handle.net/10779/uos.26135407.v1 (accessed on 2 September 2024).
47. Pan, W.; Li, X.; Tan, H.; Xu, J.; Li, X. Improvement of RTT fairness problem in BBR congestion control algorithm by gamma correction. *Sensors* **2021**, *21*, 4128. [CrossRef]
48. Njogu, C.K.; Yang, W.; Njogu, H.W.; Bosire, A. BBR-With Enhanced Fairness (BBR-EFRA): A new enhanced RTT fairness for BBR congestion control algorithm. *Comput. Commun.* **2023**, *200*, 95–103. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4123862 (accessed on 28 July 2024). [CrossRef]
49. Raiciu, C. Coupled Congestion Control for Multipath Transport Protocols. IETF RFC 6182. 2011. Available online: https://www.rfc-editor.org/info/rfc6182 (accessed on 2 September 2024).
50. Khalili, R.; Gast, N.; Popovic, M.; Boudec, J.-Y.L. MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution. *IEEE/ACM Trans. Netw.* **2013**, *21*, 1651–1665. [CrossRef]

51. Chen, K.; Shan, D.; Luo, X.; Zhang, T.; Yang, Y.; Ren, F. One rein to rule them all: A framework for datacenter-to-user congestion control. In Proceedings of the 4th Asia-Pacific Workshop on Networking, Seoul, Republic of Korea, 3–4 August 2020; pp. 44–51.
52. 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 Proceedings of the Internet Measurement Conference, Amsterdam, The Netherlands, 21–23 October 2019; pp. 130–136.
53. Quevedo Caballero, E.; Donahoo, M.; Cerny, T. Fairness Analysis of Deep Reinforcement Learning based Multi-Path QUIC Scheduling. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, 27–31 March 2023; pp. 1772–1781.
54. Ming, F.; Gao, F.; Liu, K.; Zhao, C. Cooperative modular reinforcement learning for large discrete action space problem. *Neural Netw.* **2023**, *161*, 281–296. [CrossRef] [PubMed]
55. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjel, A.K.; Ostrovski, G.; et al. Human-level control through Deep Reinforcement Learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
56. Van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
57. Bellemare, M.G.; Dabney, W.; Munos, R. A distributional perspective on reinforcement learning. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 July 2017; PMLR: Birmingham, UK, 2017; pp. 449–458.
58. Vardoyan, G.; Hollot, C.V.; Towsley, D. Towards stability analysis of data transport mechanisms: A fluid model and its applications. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1730–1744. [CrossRef]
59. Wang, Z.; Ni, H.; Han, R. Copa-ICN: Improving Copa as a Congestion Control Algorithm in Information-Centric Networking. *Electronics* **2022**, *11*, 1710. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.