# Flecto: Cross-Layer Adaptive Congestion Control with Reinforcement Learning

Cristiano Serra*†, Emilio Paolini‡, Alessio Sacco*, Roger Immich†§, Guido Marchetto*, Flavio Esposito†
† *Department of Computer Science*, Saint Louis University, USA
* *Department of Control and Computer Engineering*, Politecnico di Torino, Italy
‡ *TeCIP Institute, Scuola Superiore Sant'Anna, Italy*
§ *Digital Metropolis Institute, Federal University of Rio Grande do Norte (UFRN), Brazil*

***Abstract*—Effective congestion control is critical for wireless networks, where rapidly varying channel conditions and diverse traffic demands can severely degrade performance. Traditional congestion control algorithms rely on static heuristics that are often ill-suited for dynamic wireless environments. In this paper, we introduce Flecto, a Reinforcement Learning (RL)-based congestion control solution integrated into the QUIC protocol that, leveraging cross-layer metrics, including Signal-to-Noise Ratio, Block Error Rates, and Round-Trip Time measurements, can take decisions using a comprehensive view of network conditions. We implemented Flecto on a 5G testbed using OpenAirInterface and ETTUS USRP B210 radios, showing how it adapts transmission rates in real-time to maximize throughput and minimize latency while maintaining stability. Experimental results show that Flecto achieves an average throughput of $4539.5$ KB/s approximately $6\%$ higher both than Cubic ($4267.2$ KB/s) and New Reno ($2674.1$ KB/s) while reducing the average Round-Trip Time to $21.8$ ms, significantly lower than Cubic's $27.6$ ms and New Reno's $174.9$ ms. These performance gains underscore the promise of integrating RL with cross-layer feedback for adaptive, efficient congestion control in next-generation wireless networks. Moreover, the modular design of Flecto facilitates its extension to other transport protocols and multi-user scheduling frameworks, paving the way for broader adoption in future wireless systems.**

***Index Terms*—Congestion Control, Reinforcement Learning, Wireless Networks, Machine Learning**

## I. Introduction

Wireless networks are fundamental to modern communication, supporting applications ranging from high-speed mobile broadband to IoT and smart infrastructure [1]. The dynamic nature of wireless environments, characterized by fluctuating signal strength, interference, and user mobility, poses significant challenges for network optimization [2]. Among these, congestion control remains a critical yet complex task, as it requires dynamic management of data transmission to balance throughput, latency, and fairness under unpredictable conditions [3].

Traditional congestion control algorithms such as Transmission Control Protocol (TCP) Cubic [4] and Reno [5] rely on predefined heuristics that adapt poorly to wireless environments. These methods are designed for stable, wired networks and often struggle to accommodate the rapid variations in link quality and resource availability inherent to wireless systems. Their reliance on high-level metrics, such as packet loss or Round-Trip Time (RTT), offers only a partial view of the network, leading to inefficiencies in dynamic scenarios [6].

Machine learning (ML), particularly Reinforcement Learning (RL), offers a novel approach to congestion control by enabling data-driven, adaptive decision-making. By learning from the surrounding environment, RL-based algorithms optimize their actions based on real-time feedback. While several RL-based congestion control protocols have been proposed [7]–[12], they fail to adapt to typical wireless traffic and network conditions, which can change dramatically due to user mobility and interference.

To this end, we introduce *Flecto*, a novel RL–based congestion control algorithm that integrates cross-layer metrics—from the physical, MAC, to transport layers—to intelligently optimize network performance. By combining low-level metrics, such as Signal-to-Noise Ratio (SNR) and modulation schemes, with high-level metrics like RTT and bandwidth utilization, the proposed approach provides a complete understanding of network conditions. This comprehensive view enables the RL model to make more informed and effective decisions, optimizing performance under various scenarios.

In addition, these decisions are embedded in the QUIC protocol, which is increasingly adopted due to its superior performance compared to traditional transport protocols [13], [14]. By embedding Flecto's RL-based congestion control within QUIC, the protocol can dynamically adjust transmission rates using real-time cross-layer metrics, enhancing throughput and reducing latency in wireless environments.

In summary, the main contributions of this paper are: (i) the development of an RL-based congestion control algorithm; (ii) the comprehensive integration of cross-layer metrics to enhance QUIC decision-making; (iii) the implementation on a real-world 5G testbed using OpenAirInterface (OAI) and ETTUS USRP B210 radios; and (iv) the demonstration of significant performance improvements in throughput and latency compared to traditional methods.

The remainder of this paper is organized as follows. Section II reviews related work, highlighting the limitations of existing approaches and the potential of RL for congestion control. Section III details the design ideas behind Flecto, including the RL model design and the metrics used. Section IV describes the experimental setup and testbed configuration, analyzing the results, and comparing the proposed approach to conventional algorithms. Finally, Section V concludes the paper and outlines future research directions.

## II. Related Work

Congestion control is an important aspect of network management, ensuring data is transmitted efficiently and reliably across a network. Over the years, several methods have been developed to address congestion in various network environments.

Notable, traditional congestion control mechanisms – such as Cubic [4], Reno [5], NewReno [15], Vegas [16], and BBR [17] – have been primarily implemented at the transport layer, with TCP being one of the most widely used protocols. While these traditional congestion control algorithms have been instrumental in managing network traffic, they encounter specific limitations in wireless environments. Indeed, by relying on hard-coded rules and predefined thresholds, they fail to interpret stochastic data loss, leading to suboptimal performance when network conditions change rapidly, as is often the case in wireless networks.

The application of ML and RL to congestion control represents a significant advancement over traditional methods. Unlike conventional algorithms, ML and RL approaches can learn and adapt to dynamic network conditions, making them particularly suitable for complex and variable environments such as wireless networks [8]–[10], [18]. In this context, Aurora [10] employs a Neural Network (NN) to predict the optimal sending rate based on observed network conditions such as latency and packet loss. Similarly, Owl [8] and Orca [9] extend the Cubic implementation by adding an RL learning model that can work even in unexplored conditions. Remy [19] is set as a supervised learning technique that generates sending rules based on historical network data and predefined objectives. Sage [11], instead, combines predictive modeling with real-time data analysis by using ML techniques to forecast network congestion and using RL to adjust data transmission rates accordingly.

While these solutions show the potential of data-driven approaches in developing more adaptive and efficient congestion control algorithms,they base their decision on high-level measurements, having only a partial view of the network state. Indeed, they overlook physical layer measurements that can be very useful for adapting to the dynamic conditions of wireless networks. In addition, by modifying the TCP protocol, they inherit known limitations of this protocol when handling packet loss [3].

Solutions specifically designed for wireless and operating at different levels exist, e.g., Sprout [20] and Verus [21]; however, they are tailored to pre-determined network conditions and fail to generalize over unseen ones. In contrast, Flecto leverages cross-layer metrics within a protocol spanning layers 4 to 7, such as QUIC, enabling proactive congestion management through a more adaptive and context-aware approach.

## III. Flecto Design Overview

This section introduces the design of Flecto, our RL-based congestion control algorithm that is able to adapt dynamically to the varying conditions of wireless networks. Flecto architecture, depicted in Fig. 1, comprises three main components detailed in the following: metric collection, RL-based decision-making, and action execution.

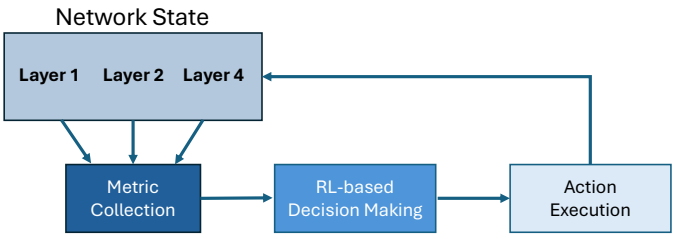



Fig. 1: System architecture of Flecto.

### A. Metric Collection

The *Metric Collection* continuously gathers metrics from different network stack layers. Unlike traditional methods that follow predefined algorithmic rules, Flecto dynamically adapts by leveraging cross-layer metrics from the physical, MAC, and transport layers. This allows gaining a comprehensive understanding of the network state, improving responsiveness to rapidly changing conditions. At the *Physical Layer (Layer 1)*, low-level metrics such as signal strength, modulation and coding schemes (MCS), and noise power are collected, offering insights into the real-time condition of the wireless channel, including signal quality and interference levels. At the *MAC Layer (Layer 2)*, the system captures medium-access metrics like Block Error Rate (BLER), scheduling decisions, and retransmission attempts, which are essential for assessing link-layer efficiency and resource utilization. At the Transport Layer (Layer 4), high-level metrics such as RTT, congestion window (`cwnd`) size, and bandwidth utilization are monitored, reflecting the overall network performance and end-to-end behavior. This complete view enables Flecto to make informed, data-driven decisions (continuously adjusting to real-time network conditions) while effectively balancing throughput and latency. Flecto employs 41 metrics collected from Layer 1, the MAC layer, and Layer 4, with 18 metrics from Layer 1, 19 from Layer 2 (MAC), and 4 from Layer 4, as detailed in Table I. Data are gathered at 1-second intervals, and to mitigate differences in logging caused by driver implementations, each sample is computed as the average of the metrics collected over the last second. Missing values are replaced with the most recent sample. These averaged metrics are subsequently fed into the RL algorithm.

### B. RL-based Model

Reinforcement Learning is a machine learning paradigm where an agent learns optimal decision-making strategies by interacting with an environment and receiving feedback in the form of rewards. Unlike traditional supervised learning, RL does not rely on labeled data but instead optimizes its actions to maximize long-term rewards through trial-and-error exploration. In the RL-based decision-making component, an RL agent processes collected network metrics to determine optimal congestion control actions. The agent's policy is

TABLE I: Cross-layer metrics used in our model

| Layer | Metrics |
|---|---|
| L1 (Physical Layer) | Blacklisted Physical Resource Blocks (PRBs), Total PRBs, Max IO, Min IO, Avg IO, Physical Random Access Channel (PRACH) IO, Current Quadrature Modulation (QM) Downlink (DL), Current Rank Indicator (RI) DL, Total Bytes TX, Uplink Shared Channel Power (ULSCH) Power, ULSCH Noise Power, Sync Pos, Round Trials, Discontinuous Transmission (DTX), Current QM UL, Current RI UL, Total Bytes RX, Total Bytes Scheduled |
| L2 (MAC Layer) | PH, PCMAX, Reference Signal Received Power (RSRP), meas, UL RI, Transmission Precoding Matrix Indicator (TPMI), Dlsch Rounds, Dlsch Errors, PUCCH0 DTX, Block Error Rate Downlink (BLER) DL, Modulation and Coding Scheme (MCS) DL, Dlsch Total Bytes, Ulsch Rounds, Ulsch DTX, Ulsch Errors, BLER UL, MCS UL, Ulsch Total Bytes Scheduled, Ulsch Total Bytes Received |
| L4 (Transport Layer) | Bandwidth Average, Congestion Window Average, Smoothed RTT, RTT Mean Deviation |

trained to maximize a reward function that captures Key Performance Indicators (KPIs) such as throughput and latency. By continuously refining its policy through interactions with the network environment, the RL agent adapts to varying traffic conditions and enhances overall network performance. In detail, at each iteration, the agent receives the current state and the reward from the dynamic system and outputs an action that optimizes a given objective. Thus, state and reward are the values that the agent receives from the system, whereas the action is the only input that the system acquires from the agent. A reward value indicates the success of the agent's action decisions, and the agent learns which actions to select to provide the highest accumulated reward over time, i.e., the long-term revenue.

In Flecto, the *state* space is constituted by Table I. The *action* space's choice is a crucial aspect in the design of an RL agent, as it determines the range of possible adjustments the agent can make, shaping its ability to respond to network dynamics efficiently. A larger action space offers greater flexibility and finer control, enabling the agent to discover optimal solutions and adapt to complex scenarios. However, this flexibility comes at the cost of increased learning complexity, longer training times, and higher computational demands. Balancing exploration and exploitation also becomes more challenging, particularly in resource-constrained environments. Therefore, selecting an appropriate action space involves a trade-off between flexibility and efficiency. After extensive experimentation, a discrete action space composed of 11 actions (as summarized in Table II) has been chosen, as it provides a favorable trade-off.

These values were selected to enable rapid and adaptive congestion window adjustments, specifically, to quickly decrease (see actions 0 and 1) and to slightly increase the latter. This responsiveness is especially beneficial in two key scenarios: *(i)* during network startup, where a fast ramp-up in bandwidth utilization is crucial, and *(ii)* in the event of packet loss, where an immediate reduction helps mitigate congestion. The exact parameters were determined after a thorough analysis of data collected from established congestion control algorithms, notably Cubic [4] and NewReno [15]. By aligning with their proven performance characteristics, our approach ensures both stability and efficiency in dynamically adapting to network conditions.

TABLE II: Actions Space

| Action Index | Action Definition |
|---|---|
| 0 | $f(x) = \frac{x}{2}$ |
| 1 | $f(x) = \frac{x}{2}$ |
| 2 | $f(x) = x - 10$ |
| 3 | $f(x) = x - 7$ |
| 4 | $f(x) = x - 3$ |
| 5 | $f(x) = x$ |
| 6 | $f(x) = x + 3$ |
| 7 | $f(x) = x + 7$ |
| 8 | $f(x) = x + 10$ |
| 9 | $f(x) = x \times 2$ |
| 10 | $f(x) = x \times 3$ |

We then design a *reward* function that encourages the agent to maximize throughput while minimizing latency, aligning with the overall goal of congestion control. We formalize this objective in the following equation:

$$R = \frac{\text{Bandwidth}}{\text{RTT}} \tag{1}$$

On one hand, maximizing throughput is essential for making efficient use of the available network capacity, allowing for the rapid transmission of data. On the other hand, reducing RTT is critical for maintaining network responsiveness and ensuring a smooth user experience. By formulating the reward as the ratio of bandwidth to RTT, the function inherently promotes configurations that achieve high data transfer rates while keeping delays to a minimum. Similar reward formulations have been employed in studies such as [22] on resource allocation in vehicular networks. Although alternative reward functions could incorporate additional network performance metrics, this formulation was chosen for its simplicity, interpretability, and proven effectiveness in dynamic wireless environments.

In addition, to deal with the large state and action spaces, we approximate the traditional RL model via NN, reducing the total available actions. This technique is referred to as Deep Reinforcement Learning (DRL) and, specifically, deep Q-Learning, which uses neural networks parameterized by $\theta$ to approximate the Q-function. Moreover, as it can happen that this approach diverges due to dynamical and frequent changes in the target [23], we also introduce a separate network, the target network. This approach is usually denoted in the literature as Deep Q-Network (DQN), and we configure a periodic update of the target network with the current Q-function. This DQN-based approach is known for its robustness and

computational efficiency, making a good candidate for our use case, where for example GPU-equipped servers are not always present. The chosen neural network consists of a Multi-Layer Perceptron (MLP) with 2 layers, each containing 64 neurons.

A key challenge during implementation was efficiently executing the training process given RL's sequential nature, which demands real-time feedback from the antennas. Unlike static data collection, real-time training requires that incoming metrics are processed and fed to the model immediately to enable adaptive learning. To address this, we implemented a real-time data pipeline integrated within the OAI. This system buffers and averages cross-layer metrics at one-second intervals while ensuring minimal latency so that the RL agent receives prompt updates from the base stations. During model training, the network conditions were carefully replicated from the data collection phase to account for the inherent variability and unpredictability of the environment.

We implement such a model using Stable Baselines3 as framework [24]. These are a set of reliable implementations of RL algorithms in PyTorch.

### C. Action Execution

The *Action Execution* component is responsible for applying the actions selected by the RL agent, serving as the system's interface for interacting with the environment. Its implementation is greatly facilitated by QUIC's presence in the user plane, which simplifies integration and deployment. Different from other TCP-based solutions, such as [8], [9], [11], this approach offers greater flexibility and adaptability to network events. This component plays a critical role, especially during deployment, as it is the only one capable of directly influencing the environment. By ensuring seamless execution of selected actions, it bridges the gap between decision-making and real-world impact, making it essential for the system's overall effectiveness.

In summary, Flecto offers a powerful approach to enhancing network performance and adaptability, particularly in the context of dynamic congestion control. By leveraging RL, Flecto continuously optimizes congestion control parameters in real-time, responding to fluctuating network conditions to maximize throughput, minimize latency, and ensure stability. This adaptive capability positions Flecto as a robust solution for next-generation wireless networks.

## IV. Testbed Implementation and Evaluation

The evaluation of congestion control algorithms presents several significant challenges. First, congestion control performance relies on multiple interdependent metrics, including throughput, latency, jitter, and packet loss. Achieving a balanced evaluation is particularly difficult, as improving one metric often comes at the expense of another. Furthermore, these factors can vary significantly over time and across different environments. Lastly, reproducibility poses a considerable obstacle as variations in test environments, traffic patterns, and network configurations make it challenging to ensure that evaluation results can be consistently reproduced across different studies.

To rigorously address these challenges, we have developed a comprehensive evaluation framework. The following subsection outlines the design and the implementation of our testbed, details the methodology used to assess Flecto's performance, and also explains the rationale behind selecting specific metrics.

### A. Testbed setup and data collection

Data was collected by exploiting the OAI framework, a versatile and open-source platform that enables the simulation, development, and testing of Next Generation wireless networks.

The testbed, as depicted in Fig. 2, is based on two separate PCs, a laptop functioning as the UE and a workstation that runs both the gNodeB and the core network software. The workstation is equipped with an Intel(R) Xeon(R) Gold 6312U CPU @ 2.40GHz with 24 cores and 64GB of ECC RAM, while the laptop features a 12th Gen Intel(R) Core(TM) i7-1260P with 12 cores, and an integrated GPU. Concerning the radio equipment and channel, the ETTUS USRP B210 Software Defined Radio module and the 5G N78 frequency are used. The adoption of real 5G radios in experimentation is important for achieving accurate and reliable results, offering several key advantages over emulated scenarios. In fact real-world 5G deployments provide a more precise representation of the complex interactions between network components, such as latency variations, signal interference, and bandwidth fluctuations, which are difficult to replicate in a simulated environment. Additionally, real 5G radios enable testing under authentic conditions, allowing for a better understanding of the performance and limitations of algorithms or applications in practical use cases.

Two different QUIC and HTTP/3 implementations are considered: a Python-based called aioquic [25] and a Rust-based called QUICHE [26], both developed by Cloudflare.

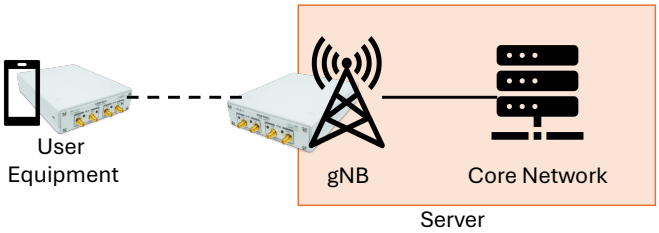



Fig. 2: The hardware setup.

To better organize data collection, the OAI's driver is configured to log the data to two different files, one for the Layer 1 metrics and one for the MAC Layer metrics. The sampling frequency is limited to once per second to mitigate the impact of potential overhead due to logging. Finally, random requests ranging from 500 KB to 500 MB are used in order to ensure proper variability in requests while providing a simple way to reproduce the results.

TABLE III: Throughput and Round Trip Time: average values and standard deviations.

| Algorithm | Throughput [KB/s] | RTT [ms] |
|---|---|---|
| New Reno | 2674.1 ± 1123.0 | 174.9 ± 182.6 |
| Cubic | 4267.2 ± 876.2 | 27.6 ± 23.2 |
| Flecto | 4539.5 ± 950.3 | 21.8 ± 1.1 |

### B. Performance Assessment

In order to evaluate the performance of our congestion control solution, we compare it with the two well-known state-of-the-art algorithms, namely NewReno and CUBIC. As evaluation metrics, the achieved throughput and RTT are considered for our analysis.

A key aspect to consider in this evaluation is the dynamicity of the 5G network environment. In particular, a proper physical setup of the antennas is crucial to ensure a fair comparison. In a 5G environment, the Reference Signal Received Power (RSRP) is a key performance indicator used to measure the signal strength of a cell's reference signal received by a UE, such as a smartphone or other connected devices. Given the logarithmic nature of the dBm measurement unit for RSRP, significant changes in actual power levels appear as relatively small changes in dBm values. However, these values can fluctuate slightly from second to second due to factors such as changes in air conditions or the presence of obstacles. Consequently, a complete repeatability is rarely achievable, except in some specific controlled environments. To ensure consistency across all experiments, we adjusted the antenna positions to exactly match the same RSRP value before starting each experiment. This approach provided the most reliable methodology given the limitations explained above.

Fig. 3a depicts the throughput performance over 100 timesteps for the three congestion control algorithms. Flecto achieves the highest throughput among the three, consistently reaching values between 4 and 6 Mbps with minimal fluctuations. Cubic exhibits a comparable throughput in certain regions but experiences more pronounced drops, indicating occasional performance degradation. NewReno demonstrates the lowest performance, with throughput values fluctuating around 3 to 4 Mbps and frequent dips. The reduced frequency of deep throughput drops indicates that Flecto adapts more efficiently to network conditions, leading to improved bandwidth utilization.

In Table III, we present the time-averaged throughput and standard deviation for each algorithm. The results show that Flecto achieves the highest average throughput at 4539.5 KB/s, exceeding Cubic by 272.3 KB/s and NewReno by 1865.4 KB/s, highlighting its more effective bandwidth utilization.

Fig. 3b compares the measured RTT evolution over 100 timesteps for the three different algorithms. Flecto maintains the lowest and most stable RTT, consistently hovering around 20 ms with minimal fluctuations. Cubic exhibits slightly higher RTT variations, occasionally spiking above 40 ms, reflecting its more aggressive congestion window growth. NewReno, in contrast, suffers from the highest and most volatile RTT, with a spike exceeding 300 ms, indicating its inefficient handling of congestion.

Table III summarizes the time-averaged RTT and its standard deviation for each algorithm. Notably, Flecto achieves a low average RTT of 21.8 ms, with only a 1.1 ms deviation, indicating highly stable latency. In contrast, Cubic records a slightly higher average RTT of 27.6 ms with moderate fluctuations ($\pm$23.2 ms). Meanwhile, New Reno exhibits considerably poorer performance, with an average RTT of 174.9 ms and extreme variability ($\pm$182.6 ms), underscoring its inefficiency in managing congestion.

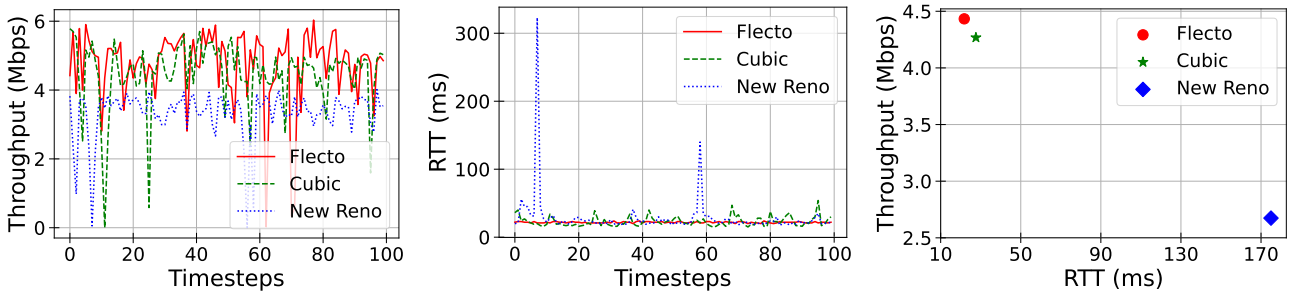
Finally, Fig. 3c provides a snapshot of each algorithm's ability to balance throughput against latency. The standard deviation is too small to be presented. Flecto (red circle) consistently achieves both a higher throughput ($\approx 4.4Mbps$) and a lower RTT ($\approx 21.8ms$) compared to the other algorithms. Cubic (green star) achieves slightly lower throughput ($\approx 4.2Mbps$) than Flecto and experiences a moderate increase in RTT ($\approx 27ms$). NewReno (blue diamond) delivers the lowest throughput ($\approx 2.7Mbps$) and the highest RTT ($\approx 174.9ms$), highlighting its limited scalability in dynamic wireless settings.

## V. Conclusion

In this paper, we introduced Flecto, a novel ML-driven congestion control approach for wireless networks, integrating cross-layer metrics into the QUIC protocol. By leveraging fine-grained measurements from the physical, MAC, and transport layers, Flecto dynamically predicts network conditions and adjusts transmission rates to improve bandwidth utilization, reduce latency, and maintain stable performance even in highly dynamic 5G environments. Our experimental evaluation on a real-world 5G testbed demonstrated that Flecto consistently outperforms traditional congestion control algorithms such as Cubic and NewReno. Notably, Flecto achieved higher throughput and more consistent low-latency performance, underscoring the benefits of integrating ML techniques with comprehensive network state information.

Despite these promising results, several challenges remain open. First, the interpretability of RL-driven decisions is an ongoing concern for network operators who require transparent and explainable systems before deploying black-box solutions. Second, while Flecto's modular design offers potential for extensibility, scaling it to multi-user and multi-flow scenarios in heterogeneous wireless environments introduces significant challenges in terms of model complexity and real-time data processing.

Overall, Flecto marks a significant advancement in adaptive and efficient congestion control for wireless networks. Its modular design can be extended to other transport protocols and multi-user scheduling frameworks, broadening its applicability across diverse networking scenarios. As wireless standards evolve and device capabilities advance, RL-driven congestion control schemes are poised to play an increasingly central role in ensuring high performance, reliability, and responsiveness in next-generation networks. Flecto's approach paves the way for more intelligent, data-driven network optimization, reinforcing




(a) Throughput vs. Time (b) RTT vs. Time (c) Throughput vs. RTT

Fig. 3: Comparison between Flecto, Cubic, and New Reno. Flect can jointly maintain high throughput and low delay over time.

the potential of machine learning in shaping the future of wireless communications.

## ACKWNOWLEDGEMENT

This work has been supported by the National Science Foundation (NSF) Award OAC # 2201536, and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This study is also supported by the Chips Joint Undertaking (JU), European Union (EU) HORIZON JU-IA, and its members (including top-up funding by MIMIT), under grant agreement No. 101140087 (SMARTY), and from the NGI Enrichers program, funded by the European Union's Horizon Europe Research and Innovation Programme under grant agreement 101070125.

## REFERENCES

[1] A. Gupta and R. K. Jha, "A survey of 5g network: Architecture and emerging technologies," *IEEE access*, vol. 3, pp. 1206–1232, 2015.

[2] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao *et al.*, "A variegated look at 5g in the wild: performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 610–625.

[3] J. Lorincz, Z. Klarin, and J. Ožegović, "A comprehensive overview of tcp congestion control in 5g networks: Research challenges and future perspectives," *Sensors*, vol. 21, no. 13, p. 4510, 2021.

[4] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[5] Reno explanation. [Online]. Available: https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/

[6] C. Luo, F. R. Yu, H. Ji, and V. C. Leung, "Cross-layer design for TCP performance improvement in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 5, pp. 2485–2495, 2010.

[7] A. R. Andrade-Zambrano, J. P. A. León, M. E. Morocho-Cayamcela, L. L. Cárdenas, and L. J. de la Cruz LLopis, "A reinforcement learning congestion control algorithm for smart grid networks," *IEEE Access*, 2024.

[8] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Owl: Congestion Control with Partially Invisible Networks via Reinforcement Learning," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–9.

[9] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual conference of the ACM Special Interest Group (SIGCOMM '20)*. ACM, 2020, pp. 632–647.

[10] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 3050–3059.

[11] C.-Y. Yen, S. Abbasloo, and H. J. Chao, "Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control," in *Proceedings of the ACM SIGCOMM 2023 Conference*. ACM, 2023, pp. 255–274.

[12] L. Pappone, A. Sacco, and F. Esposito, "Mutant: Learning Congestion Control from Existing Protocols via Online Reinforcement Learning," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '25')*. USENIX Association, 2025.

[13] J. Iyengar, M. Thomson *et al.*, "QUIC: A UDP-based multiplexed and secure transport," in *RFC 9000*. Internet Engineering Task Force (IETF) Fremont, CA, USA, 2021.

[14] M. Soni and B. S. Rajput, "Security and performance evaluations of quic protocol," in *Data Science and Intelligent Applications: Proceedings of ICDSIA 2020*. Springer, 2021, pp. 457–462.

[15] S. Floyd, T. Henderson, and A. Gurtov, "The newreno modification to tcp's fast recovery algorithm," Tech. Rep., 2004.

[16] L. S. Brakmo and L. L. Peterson, "Tcp vegas: A fair and efficient congestion control algorithm," *IEEE Journal on selected areas in communications*, 1995.

[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[18] H. Tian, X. Liao, C. Zeng, J. Zhang, and K. Chen, "Spine: an efficient drl-based congestion control with ultra-low overhead," in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22)*. ACM, 2022, pp. 261–275.

[19] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 123–134, 2013.

[20] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, 2013, pp. 459–471.

[21] Y. Sakai, T. Aalto, R. Chandra, and M. Steiner, "Adaptive Congestion Control for Unpredictable Cellular Networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015, p. 127–138.

[22] H. Ye and G. Y. Li, "Deep reinforcement learning for resource allocation in v2v communications," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[23] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[24] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[25] aiortc, "aioquic: An implementation of the quic protocol in python," January 2025, accessed: 2025-01-13. [Online]. Available: https://github.com/aiortc/aioquic

[26] Cloudflare, "Quiche: A quic implementation by cloudflare," January 2025, accessed: 2025-01-13. [Online]. Available: https://github.com/cloudflare/quiche