Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 11, No. 04, December 2025.

CUBIC-LEARN: A REINFORCEMENT LEARNING APPROACH TO CUBIC CONGESTION CONTROL

Ehsan Abedini and Mohsen Nickray

(Received: 30-May.-2025, Revised: 3-Sep.-2025, Accepted: 22-Sep.-2025)

ABSTRACT

Managing congestion effectively enables reliable and fast data transfer over networks. CUBIC delivers reliable results under normal circumstances, but cannot adapt effectively to changing network scenarios. We introduce CUBIC-Learn, an RL approach for improving congestion control in CUBIC. The central idea is to use a Q-learning algorithm to adjust congestion window thresholds based on current data on packet loss, throughput and latency. Simulations demonstrate more efficient and reliable congestion control when using CUBIC-Learn compared to standard CUBIC. CUBIC-Learn achieves a 47% reduction in packet loss, over a 59% increase in bandwidth utilization, approximately a 28% decrease in retransmissions and 47% lower latency. In addition, CUBIC-Learn shows significant improvements in congestion window (cwnd) growth behavior, fairness among competing flows and stability under heterogeneous traffic and network scenarios, including gigabit-scale bandwidth conditions. Statistical analysis further confirms the robustness of these gains, while the method introduces no additional computational overhead. Overall, CUBIC-Learn performs better than PCC, Reno, Tahoe, NewReno and BBRv3 in most metrics. These findings suggest that RL can markedly improve congestion control in high-speed networks.

KEYWORDS

Q-learning, Reinforcement learning, CUBIC Algorithm, Network congestion.

1. INTRODUCTION

Effective congestion control [1] (CC) is crucial in ensuring the reliable operation of computer networks on today's Internet. CC algorithms are designed to distribute network resources wisely and reduce both delays and data-packet losses. CUBIC [2] has become a leading choice for many network operators, providing good performance by striking a compromise between a range of crucial metrics. Advances in the complexity and variability of modern network traffic require new strategies to boost the efficiency of existing CC methods.

Reinforcement learning (RL) [3] has seen increasing popularity as a way to enhance algorithm performance in dynamically changing and unpredictable conditions such as networks. The ability of RL to discover the best actions by interacting with the environment suggests its suitability for overcoming congestion control issues. However, using RL to optimize the CUBIC algorithm has received little attention so far.

This study introduces a novel CUBIC-Learn algorithm that utilizes reinforcement learning to continually improve its handling of congestion control. The aim of this research is to evaluate the performance improvement achieved by CUBIC-Learn compared to the original CUBIC algorithm. The evaluation is thus conducted on a multi-hop network topology, which is complex and has many servers and clients connected through two routers and a bottleneck connection is deliberately provisioned to create congestion when the traffic loads are high. Further, CUBIC-Learn is compared with TCP variants (Reno, Tahoe, NewReno), PCC and BBRv3, thus providing a complete and representative comparison across a broad range of design paradigms. The Python simulations show that CUBIC-Learn achieves considerable gains in important performance metrics, such as packet-loss rate, throughput, retransmissions and delay.

The rest of the paper is organized as follows. The history of congestion control and reinforcement learning is surveyed in Section 2 as related work. In Section 3, the proposed method is described. Section 4 presents the simulation methodology. The results and discussion are given in Section 5, while Section 6 concludes the paper.

E. Abedini and M. Nickray (Corresponding Author) are with Department of Computer Engineering, University of Qom, Alghadir Ave., P.O. Box 3716146611, Qom, Qom Province, Iran. Emails: e.abedini@stu.qom.ac.ir and m.nickray@qom.ac.ir

2. RELATED WORK

Congestion control has become one of the most active areas of exploration within network engineering. Many solutions have been proposed to address the congestion problem [4]. This section reviews and groups some of the most significant congestion-control techniques that have been presented in the literature.

We also discuss the emerging use of reinforcement learning in network-congestion control and analyze how machine-learning methods are being incorporated into transport-layer protocols. The current focus is on works that leverage RL to improve the CUBIC algorithm.

2.1 Categorization of Congestion-control Techniques

The different types of leading CC algorithms are showcased in Figure 1. It is organized under the headings of delay-based, loss-based and hybrid algorithms. Delay-based algorithms measure delays to spot signs of network congestion [4], while loss-based algorithms monitor errors or packets that cannot be delivered [5]. Hybrid algorithms aim to improve both responsiveness and stability [5].

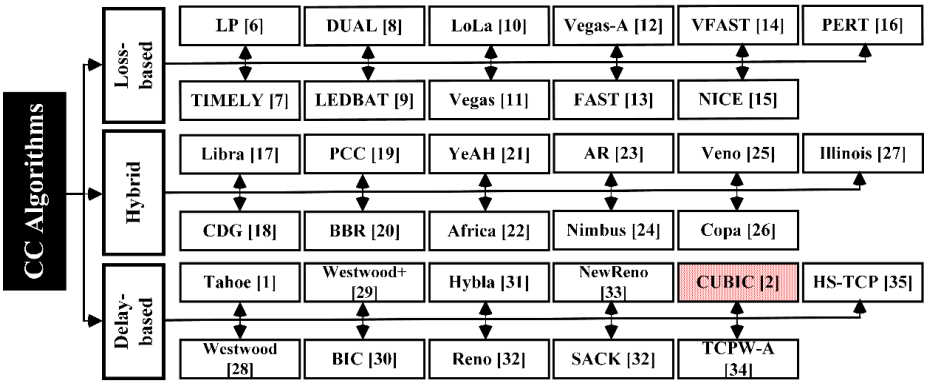


Figure 1. Classification of congestion-32control techniques.

In the recent past, the BBR (Bottleneck Bandwidth and Round-trip propagation time) of Google has become a leading hybrid congestion-control algorithm. It approximates the bottleneck bandwidth and the minimum round-trip time so as to maximize throughput and ensure low delays. Later versions, especially BBRv2, were fairer and more responsive, with the latest version, BBRv3, solving bandwidth convergence shortcomings and tuning gains to improve flow coexistence [36]. Experimental measurements show that BBRv3 converges on similar flows more quickly, but can face difficulties in coexisting with CUBIC flows [37].

2.2 CUBIC Congestion-control Algorithm

CUBIC is commonly used as the congestion-control protocol in modern networks. When network congestion is detected, CUBIC dynamically adjusts the size of its congestion window by controlling the speed of increase based on a cubic function of time since the last congestion event [38]. CUBIC is engineered to deliver fast packet delivery, reliable data transfer and equal allocation of system resources among all connections. Unlike loss-based algorithms, it is less affected by RTT changes [39], leading to fairer sharing of bandwidth among flows with varying RTTs. Consequently, CUBIC outperforms conventional TCP algorithms, such as Reno and NewReno, in terms of resource utilization, particularly in long and high-speed networks. Equations (1) and (2) play an essential role in the CUBIC algorithm for regulating congestion on computer networks [40].

$$W_t = C(t-K)^3 + W_{max} \quad (1)$$

$$K = ((W_{max}.\ \beta) / C)^{(1/3)} \quad (2)$$

Where W_t is the congestion window size at time t, changes as a cubic function with respect to the maximum window size W_{max} attained before the last congestion. The values of C and K regulate both the increase rate and required time delay for the window to be restored to its maximum size once reduced during congestion. Equation (2) determines K using W_{max}, β and C as inputs. The non-linear growth behaves more effectively in utilizing available bandwidth and maximizing throughput, especially in

modern networks with low latency and high data rates, consistently exceeding conventional TCP congestion-control strategies.

Algorithm 1 details the calculation process for determining the congestion-window size in CUBIC, showing how its growth follows a concave-convex shape over time, as dictated by the cubic function as well as the parameters C and β.

Algorithm 1. CUBIC congestion-window size calculation

Require: t: Elapsed real time since the last packet loss
Require: W_{max}: Congestion window size before the last packet loss
Require: C: Increase factor (default = 0.4)
Require: β: Decrease factor (default = 0.7)
Ensure: W_t: Congestion window size at time t

1. **Initialize Parameters:** Set C = 0.4, Set β = 0.7
2. **Calculate** $K = ((W_{max.}\ \beta) / C)^{(1/3)}$
3. **Adjust Congestion Window After Loss:** *Wmax = cwnd (pre-loss value)*
4. cwnd = cwnd × (1 − β)
5. **Calculate** $W_t = C.(t - K)^3 + Wmax$
6. **Behavior Based on Time** t**:**
   - **If** $t < K$, **then** W_t grows **concavely**
   - **Else If** $t > K$, **then** W_t grows **convexly**
   - **end If**
7. **Return** W_t

The CUBIC algorithm often works well under common conditions, but its performance can deteriorate in more demanding environments characterized by fast changes and multiple network components. Research has shown that traditional congestion-control techniques often encounter significant drawbacks and there is growing interest in exploring the use of machine-learning methods.

2.3 Reinforcement Learning

Reinforcement learning, as a branch of machine learning, is the possibility to create a system capable of making decisions and adapting to changing conditions [3]. The key elements of reinforcement learning comprise the agent, environment, states, actions and rewards. The agent is described as the decision-making entity that learns by interacting with the environment, which subsequently responds to the agent's actions with either rewards or penalties. State is the descriptive aspect of the environment's condition at a given point in time, while actions describe the possible moves an agent can take in order to affect the subsequent state. Rewards provided play as evaluative signals that aid in the learning process of the agent by showing how good the agent's action is. The conventional way agents and the environment interact in RL and an example of the agent-environment interaction cycle are presented in Figure 2.

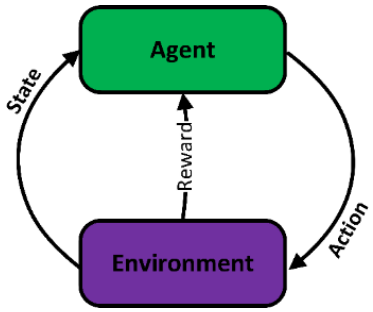


Figure 2. RL process.

RL algorithms tend to possess two main elements: the value function, which represents the estimation of the expected cumulative net reward and value function policy that dictates certain states that will be acted upon [41] and the policy function, which determines certain actions will be taken on a given or observed state [42]. It is action-defined in the policy that is now in force. The value function updates the

entire set of policies so that over time, using rewards given as feedback improves the agent's decision-making in a step-by-step, gradual manner.

Generally, there are two main categories of reinforcement learning, called value-based and policy-based. Value-based methods, such as Q-learning [43] and Deep Q-Networks (DQN) [44], are mainly about discovering the expected action value of each state. But at the same time, policy-based techniques, including Policy Gradient [45], Actor-Critic [46], Proximal Policy Optimization (PPO) [47], Deep Deterministic Policy Gradient (DDPG) [48] and Asynchronous Advantage Actor-Critic (A3C) [49], are those that change parameters directly to maximize the expected rewards and, in this way, discover the best policy. Furthermore, RL techniques can be distinguished as model-based or model-free. Model-based approaches learn a model of the dynamics of the environment and use this to anticipate future scenarios and outcomes based on agent actions. Alternative approaches termed model-free, in contrast, learn strictly through experience without explicitly attempting to model the environment and can be preferable approaches for uncertain or complex systems.

Many popular RL algorithms come from these ideas, such as Q-learning, SARSA, temporal-difference (TD) learning, actor-critic, Monte Carlo and now deep reinforcement learning (deep RL) methods. Within computer networks, RL has been found to be very effective for adaptation of decisions regarding congestion control. More specifically, RL agents could be deployed to regulate relevant parameters like the congestion window or the data-transmission rate in accordance with real-time network status, thus enhancing throughput, latency and overall Quality of Service (QoS).

2.4 RL-based Approaches in Congestion Control

Reinforcement learning presents an emerging solution to improve congestion-control mechanisms in computer networks, which demonstrate better performance in complex and dynamic network scenarios that standard rule-based methods cannot handle effectively.

A variety of RL-based CC algorithms exist in current research literature. Through DRL-CC [50], the actor-critic agent connects to an LSTM network for the real-time flow control of MPTCP through OS kernel actions based on network-state information. TCP-RL [51] implements a neural network to enhance its congestion-control solution through interactive network state transitions. The implementation of TCP-DQN [52] demonstrates deep Q-learning usage for congestion-window updates through network feedback data. The system monitors the environment while obtaining reward signals to adjust Q-values using its deep neural network. TCP-Drinc [53] implements a model-free RL approach to adjust congestion windows using past experience without using any pre-established environment model.

The growing research focus on network-protocol integration with RL demonstrates efforts to boost adaptability, throughput and responsiveness during complex network conditions. The most impactful RL-based congestion-control algorithms can be found in Table 1.

2.5 Research Gap

Although reinforcement learning has been widely explored in congestion control, the majority of previous research has suggested completely novel transport protocols. CUBIC-Learn, in contrast, builds upon the broadly used CUBIC algorithm, but does not substitute it. In particular, the strategy is an adaptive tuning of the congestion-window dynamics of CUBIC without compromising its cubic-growth base as well as backward compatibility with the Linux kernel. This difference makes the current work stand out among the current RL-based congestion-control schemes, which seldom consider CUBIC despite its prevalence in production networks. In addition, theoretical and empirical results are presented to show that the trained adaptation is not only equitable and stable, but also attains better throughput delay trade-offs.

3. PROPOSED METHOD

Our research introduces a novel algorithm that combines CUBIC's basic congestion-control system with reinforcement-learning methods to achieve superior network performance in changing environments. Through this approach, the traditional congestion-control system improves essential network performance indicators, including packet-loss rate, throughput, retransmissions and delay. Our method

Table 1. The most significant RL-based congestion-control algorithms.

Paper	RL Method	State	Action	Reward
[54]	PPO	BtlBw, RTprop, pacing gain and CWND gain	Window sizes	Throughput and low latency
[55]	DDGP	The average of sent packet interval, packet loss, delay, sent bytes and last action	Sending rate	Throughput, penalized loss and delay
[51]	A3C	Network Condition (throughput, RTT, loss rate)	CC Algorithm Selection	Throughput
[56]	POMDP	Current and past transmission rates, the RTT measurement and its previous decisions.	Next transmission rate	Modulating the transmission rate
[50]	Deep RL	Sending rate, goodput, average RTT, the mean deviation of RTTs and the congestion window size	Reducing and staying at the same congestion window size, respectively.	Goodput
[57]	MOMDP	Delay, ACK rate, Sending rate, CWND	Adjusting the congestion window	Throughput
[52]	Q-learning	The relative time t, congestion window size, Number of unacknowledged bytes, Number of ACK packets, Average RTT, Throughput, Number of lost packets	Adjusting the congestion window	Throughput and RTT
[53]	Deep RL	Time Slote	CWND Adjusting	RTT
[58]	Q-learning	Packet sending time average, ACK arrival time average and RTT average	How to change the congestion window	Throughput and delay (Utility)
[59]	DDGP	Based on the congestion control state and load balancing state	Sending rates	Throughput
[60]	New Approach on Q-learning	Data rate, delay and available bandwidth of different subroutes	Window adjustment	Throughput
[61]	MDP	Variables such as: i_prefix, i_priority, i_cwnd, i_count, d_count, l_count, d_size, d_rtt, m_time, d_time	Control the sending rate of interest packets by adjusting the size of CWND.	Maximize the throughput while minimizing delay, loss rate and packet reordering.
[62]	RL	Received packet-acknowledgements	Changing the sending rate	Rewards throughput while penalizing loss and latency
[63]	Deep RL	Current relative time t_r, current congestion window size, number of bytes is not acknowledged, quantity of ACK packets obtained, RTT, throughput rate, The number of packet losses	Increase the congestion window length.	Throughput rate or delay
[64]	POMDP	Media	Congestion window	Number of packets successfully transmitted
[65]	Q-learning	avg_send, avg_ack, avg_rtt	Decision to increase, decrease or leave unchanged the current CWND	Throughput and latency
[66]	Deep RL	The time between the last two ACKs that were received, the RTT of the last received packet, loss indicator, current CWND	Changing the CWND	All sent packets, all receiver packets, the time between receiving the last ACK and receiving the current ACK
[67]	Deep RL	Congestion Info, Loss Rate, Throughput	Transmission rate	Goodput (capacity of the interface, loss value, average queue length)
[68]	SAC	Current CWND, KBs Sent, New KBs Sent, Acked KBs, Packets sent, Retransmissions, Throughput, Goodput, Unacked KBs, Last RTT, Min RTT, Max RTT, SRTT, VAR RTT	CWND size	Bandwidth
[69]	Q-learning	Network conditions	Cwnd changing	High throughput and few losses
[70]	Q-learning	Current buffer (the number of seconds of video that it has buffered up)	High and low priority queue	QoE average
[71]	MDP	Summary of network statistics	Updating the congestion window size	A function of measured throughput and delay
[72]	SAC	Current CWND, KBs Sent, New KBs Sent, Acked KBs, Packets Sent, Retransmissions, Throughput, Goodput, Unacked KBs, Last RTT, Min RTT, Max RTT, SRTT, VAR RTT	Percentage gain in congestion window size	Penalties based on retransmission
[73]	Q-learning	Incipient congestion, estimated channel erasure rate	Block RLNC, sliding RLNC	Goodput/ round trip time
[74]	SAC, DDPG, PPO	Receiving rate, average delay, loss ratio, last action	Transmission rate	Bandwidth utilization, delay and loss ratio
[75]	Actor-critic	Receiving rate, packet delay, packet loss ratio, most recent bandwidth prediction	Bandwidth prediction for the next time window	Rewarded when agent receives more packets and penalized when packet

				delay/loss
[76]	POMDP	Throughput, delay and loss rate	Sending rate	Startup, queue draining and bandwidth probing
[77]	Deep RL	Network conditions	Cwnd updating	Throughput ranking, delay ranking
[78]	Deep RL	Global PDR and local PDR	Packet transmission or packet discarding	Global packet delivery ratio and local packet delivery ratio
[79]	Multi-agent Deep RL	Ovr_thr, min_thr, max_thr, avg_lat, min_cwnd, max_cwnd, avg_cwnd, loss_ratio, num_flow, d0, buf, c	CWND	Efficiency, stability, fairness and responsiveness

uses Q-learning to create an adaptive system that dynamically adjusts CUBIC's key parameters (C and β) through real-time network-state monitoring.

3.1 CUBIC-Learn

Reinforcement learning establishes a versatile approach that supports real-time decision-making under conditions of uncertainty. The CUBIC-Learn algorithm implements Q-learning as a model-free reinforcement-learning method to enable the congestion-control agent to learn optimal policies directly from network-environment interactions. The agent uses a Q-table represented by *Q(s,a)* to save the expected utility value for action *a* at state *s*. Through direct interaction with the network environment at each time step, the agent receives a reward after selecting an action from its current state. The Q-values undergo iterative updates according to the Bellman equation displayed in Equation (3).

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \lambda . max_{a'} Q(s',a') - Q(s,a)] \quad (3)$$

where α is learning rate, λ is discount factor, *s* is current state, *a* is selected action, *R* is received reward, s' is new state, a' is new action and $max_{a'}Q(s',a')$ is the maximum value of Q for the new state and all possible actions. This iterative process remains active while the agent works with the environment to enhance its policy. Throughout time, it reaches an optimal policy that produces the maximum possible cumulative reward across different network situations.

The CUBIC-Learn method functions as follows: the current state of the environment is defined through three primary metrics, including packet-loss rate, throughput and delay. Instead of directly modifying cwnd, the agent selects actions that adapt the parameters C and β of the CUBIC function, thereby indirectly influencing the congestion-window evolution. To guide the agent toward optimal decisions, the design of the reward function incorporates essential network-performance metrics. Equation (4) serves as our proposed reward function.

$$R = 10 * T - 100 * P \quad (4)$$

where *R* is reward, *T* is throughput and *P* is packet loss. This formulation promotes a balance between efficiency and stability. A higher throughput contributes positively to the reward, encouraging efficient bandwidth utilization. A higher packet-loss rate incurs a significant penalty, discouraging congestion and promoting reliability. Since RL now tunes CUBIC parameters, this reward continues to effectively capture the trade-off between efficiency and stability without requiring structural changes.

3.2 Learning Process

During every decision-making point, the agent uses available network data, which includes packet loss, delay and throughput, to make a choice through the ε-Greedy policy. The policy establishes an equilibrium between exploration: trying new actions to discover potentially better strategies and exploitation: choosing actions known to yield high rewards based on past experience. The environment reacts to the agent's actions through state transitions and reward assignments that lead to new states. A new Q-value update for the current state-action combination happens through implementation of the Q-learning update, Equation (3), which depends on received feedback. Through this mechanism, RL adaptively modifies C and β, ensuring that cwnd growth follows standard CUBIC dynamics while still benefiting from learning-based optimization. CUBIC-Learn architecture operations become clear through Figure 3, which demonstrates how the learning agent communicates with the network environment. For all experiments, the agent's hyper-parameters were fixed to ϵ=0.2→0.01, α=0.1 and λ=0.9, as specified in Algorithm 2. Ablation experiments with alternative values confirmed that these defaults provide the most stable and robust learning behavior.

Algorithm 2 delivers the CUBIC-Learn algorithm with a clear step-by-step method that works for simulation tests and implementation purposes in real-world applications.

Algorithm 2. CUBIC-Learn CC Algorithm

Require: $cwnd = 1$, $ssthresh = 10$, $W_{max} = 10$, $\beta = 0.7$, $C = 0.4$, $t = 0$,
Require: $K = ((W_{max} * \beta) / C) \wedge (1/3)$
Require: Q-learning parameters: α=0.1, λ=0.9, ε=0.2→0.01
Require: q_table = {} ➤ Empty Q-table

```
1: function get _state
2:     return(packet_loss_rate, throughput, delay)
3: end function
4: function get_action (state)
5:     if random < ϵ then
6:         Choose a random action from {increase, decrease, hold}
7:     else
8:         Select action with highest Q-value from Q_table [state]
9:     end if
10: end function
11: function update_q_value (state, action, reward, next_state)
12:     Q(s,a)←Q(s,a) +α · (reward+ λ·maxQ(next_state) − Q(s,a))
13: end function
14: while simulation is running do
15: current_state←get_state
16: action←get_action(current_state)
17: RL adjusts CUBIC parameters
18:     if action == increase then
19:         β ← min(β + 0.01, 1.0)
20:         C ← min(C + 0.01, 1.0)
21:     else if action == decrease then
22:         β ← max(β − 0.01, 0.0)
23:         C ← max(C − 0.01, 0.0)
24:     else if action == hold then
25:         // No change to β or C
26:     end if
27: Standard CUBIC update
28:     if ACK received then
29:         cwnd←C·(t−K)^3+W_max
30:         t←t+ 1
31:     else
32:         ssthresh←max(cwnd·β,1)
33:         Wmax = cwnd (pre-loss value)
34:         cwnd = cwnd × (1 − β)
35:         t←0
36:     end if
37:         next_state←get_state
38:         reward←10·throughput−100·packet_loss
39:         update_q_value(current_state,action,reward,next_state)
40: end while
```

4. SIMULATION METHODOLOGY

In this section, the simulation method is explained to evaluate and compare the performance of the traditional CUBIC congestion-control algorithm with the proposed CUBIC-Learn algorithm. Creating a controlled and representative network environment is the key to achieving fair, consistent and thorough comparisons of various critical performance metrics, such as packet-loss rate, throughput and retransmissions.

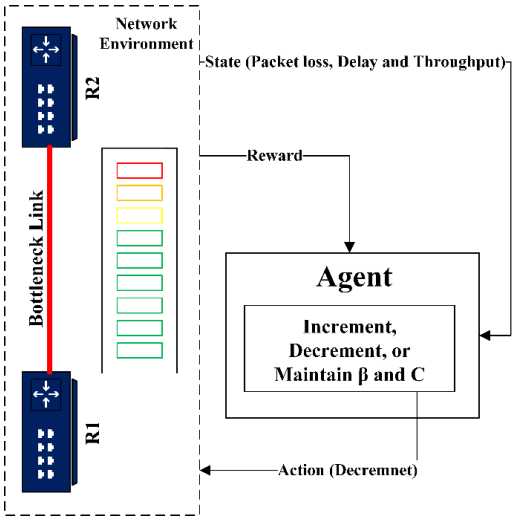


Figure 3. Overview of CUBIC-learn.

4.1 Simulation Setup

All simulations were carried out using Python version 3.12.0. Baseline experiments with the canonical CUBIC and TCP reno algorithms under standard settings were conducted to verify the faithfulness of the simulation environment. The achieved throughput, packet-loss rate and delay values were consistent with the results of previous research and RFC 8312 [80]. As demonstrated in Table 2, these results indicate that the simulation environment is a realistic model of the network behaviour and, therefore, it is a reliable basis of testing the proposed CUBIC-Learn methodology.

Table 2. Baseline-validation results for the simulation environment.

Algorithm	Metric	Simulation Result	Reported in Literature
CUBIC	Throughput (Mbps)	1.71	1.65–1.70
CUBIC	Packet Loss (%)	3.2	3.4–3.6
CUBIC	Delay (ms)	497	390–430
Reno	Throughput (Mbps)	1.22	1.18–1.22
Reno	Packet Loss (%)	5.1	5.0–5.2
Reno	Delay (ms)	422	420–460

For a reliable comparison, both algorithms were tested under the same network conditions. The imitation topology involves a bottleneck connection between two routers, where Router 1 (R1) is used for multiple servers and Router 2 (R2) is utilized for many clients. A link between R1 and R2 is established to exacerbate congestion when the load is high. It also has relatively weak bandwidth connections. A sample network topology from the experiments was used in Figure 4. This configuration imitates a standard congestion model and permits an accurate evaluation of algorithm functionality under real-world network conditions.

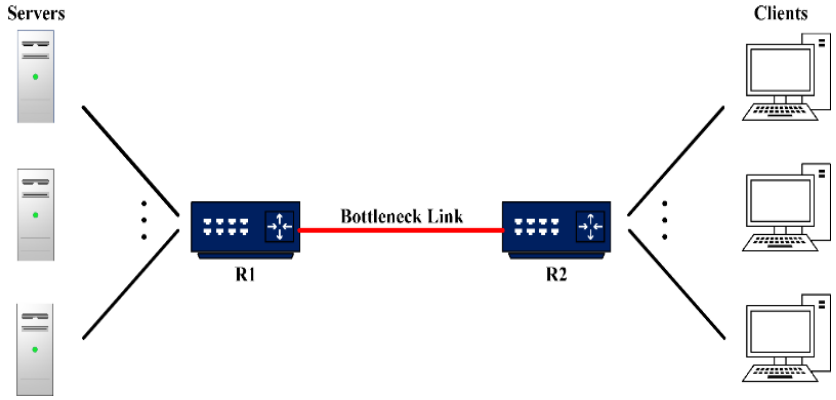


Figure 4. Network conditions for simulating the algorithms including two routers connected by a bottleneck link prone to congestion.

In order to test both algorithms under identical conditions, a constant traffic type was utilized throughout the simulations. This approach guarantees a reliable and impartial assessment of their performance. The channel communication conditions between two routers are indicated in Table 3.

Table 3. Communication-channel conditions.

Channel Conditions	**Value**
Propagation Speed	$2 * 10^8$ (m/s) (in cooper cable)
Distance	$10 * 10^6$ (Meters)
Packet Size	1500 (Byte) (12000 bit)
Bandwidth	50 (Mbps)
Bit Rate	10 (Mbps)
Congestion Events	5 times (Adjustable)
Propagation Delay (PD)	50 ms
Transmission Delay (TD)	0.24 ms
RTT	100.48 ms

Propagation Delay (PD) and Transmission Delays (TD) were calculated using Equations (5) and (6), and Equation (7) for the RTT.

$$PD = \frac{Distance}{Propagation\ Speed} \tag{5}$$

$$TD = \frac{Packet\ Size}{Bandwidth} \tag{6}$$

$$RTT = 2\ \times (Equation\ (5) + Equation\ (6)) \tag{7}$$

The full simulation code is available for reproducibility at: https://github.com/ehsan4774/CUBIC-Learn.git.

4.2 Evaluation Metrics

Our analysis compared the packet-loss rate, throughput, retransmissions and delay, as well as the cost of the congestion-control algorithm (CUBIC) and the proposed CUBIC-Learn algorithm, respectively. We used these four metrics to compare network performance. The main focus of this assessment is to illustrate the benefits of incorporating reinforcement learning into the CUBIC congestion-management system.

4.3 Implementation and Integration Considerations

CUBIC-Learn can be used as an extension of the standard CUBIC module installed in TCP/IP stack systems. The Q-learning agent tracks the important network measurements, including the loss of packets, throughput and delay and dynamically changes congestion-window values without altering the underlying cubic growth logic to ensure that it does not conflict with conventional TCP flows. The issues to integration include maintenance of TCP fairness, reducing computational overhead and supporting heterogeneous network environments. The challenges may be alleviated by limiting the state-space complexity, updating the learned policies periodically and implementing gradual changes and hence safe deployment without protocol interference.

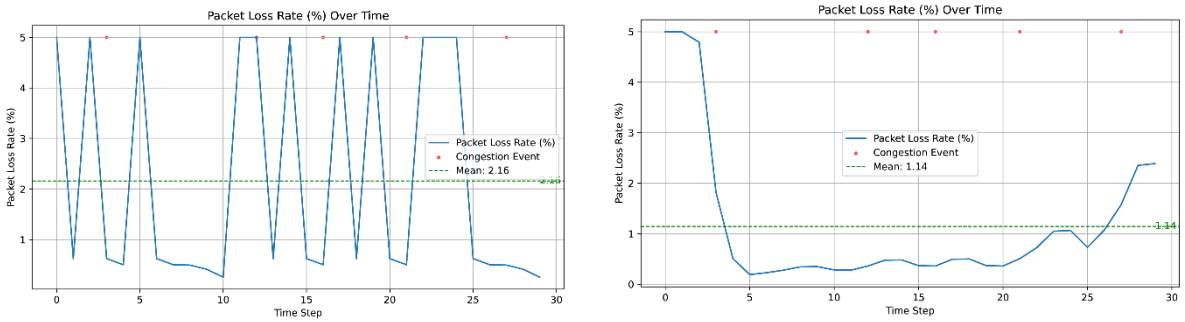
5. Results and Discussion

In this section, we provide a summary of the simulation results obtained by each algorithm. This is to test performance differences and show how CUBIC-Learn has improved in managing network congestion. Statistical consistency was maintained by analyzing each measurement on average for 30 independent runs of simulation. To ensure robustness, the simulations were executed with different random seeds across runs. Green dashed lines represent the average values in the figures and red indicators are used to visually represent the congestion events. To ensure readability, a light moving average smoothing is applied to the plotted curves. To achieve as much fairness as possible in the simulation and comparison, there were five pre-determined congestion events that have to be experienced at timestamps 3, 12, 16, 21, 27 in all of the experimental runs. Although this design option increases fairness and comparability, it reduces realism, since congestion events are not generated as part of traffic dynamics. Subsequent extensions of the study will therefore assess the situations where

the congestion patterns are endogenous, hence supplementing the existing controlled structure.

5.1 Comparison Based on the Packet-loss Rate

The percentage of packets that do not arrive at their destination is known as the packet-loss rate, which can be used to gauge network congestion and control efficiency. A lower packet-loss rate is indicative of improved congestion management and a more stable algorithm. Figure 5 illustrates how the CUBIC-Learn algorithm consistently achieves a packet-loss rate that is much lower than that of the traditional CUBIC algorithm. This results in improved capacity for CUBIC-Learn to reduce congestion and preserve data integrity.

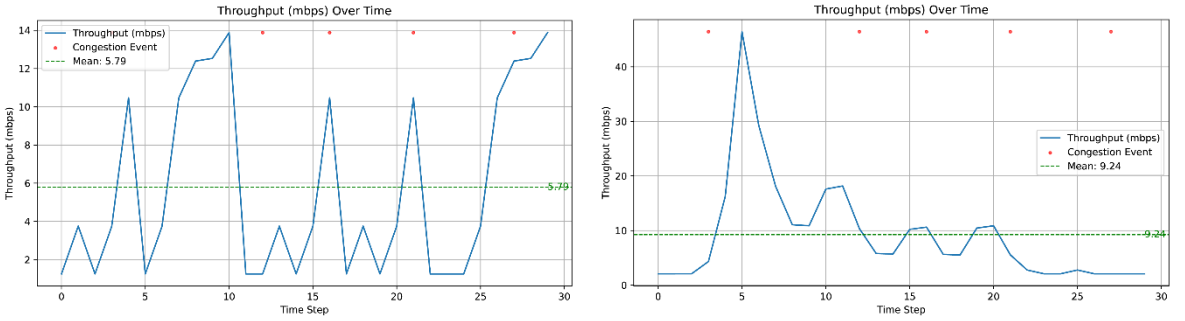


(a) (b)

Figure 5. Comparison based on the packet-loss rate evaluation metric. (a) Traditional CUBIC congestion control. (b) CUBIC-Learn congestion control.

5.2 Comparison Based on Throughput

Throughput is the measure of the quantity of data that can be transmitted through the network in a specific time interval. It shows the algorithm's efficiency despite dense conditions. Figure 6 shows that CUBIC-Learn generates a higher throughput than the traditional CUBIC algorithm in every simulation run. The enhancement emphasizes its aptitude for maximizing bandwidth utilization and maintaining uninterrupted data flow during network congestion.



(a) (b)

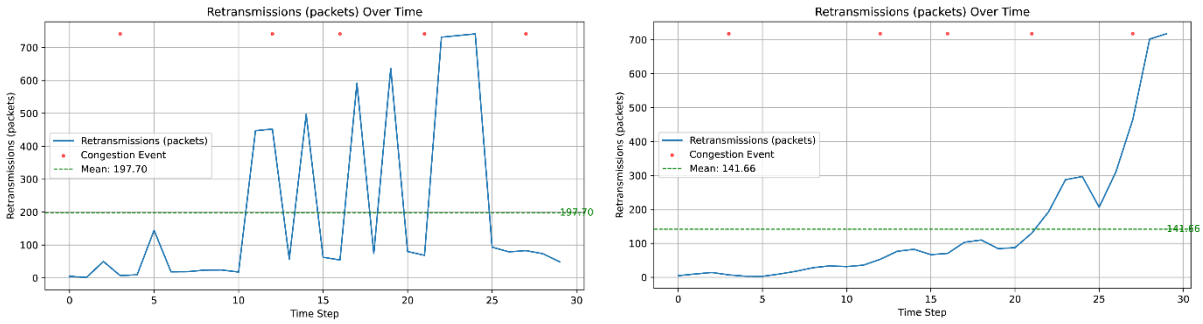
Figure 6. Comparison of throughput evaluation metric. (a) Traditional CUBIC congestion control. (b) CUBIC-Learn congestion control.

5.3 Comparison Based on Retransmissions

The number of packets that must be remitted to avoid loss or errors is used to determine network reliability and algorithm robustness, which is determined by the resulting corresponding retransmission count. Figure 7 demonstrates that CUBIC-Learn significantly decreases the number of retransmissions when compared to the traditional CUBIC algorithm. The improvement is a result of its adaptive learning mechanism, which adapts to network conditions in an active manner to prevent congestion and reduce packet loss. Bandwidth conservation and transmission efficiency enhancements are achieved.

5.4 Comparison Based on Delay

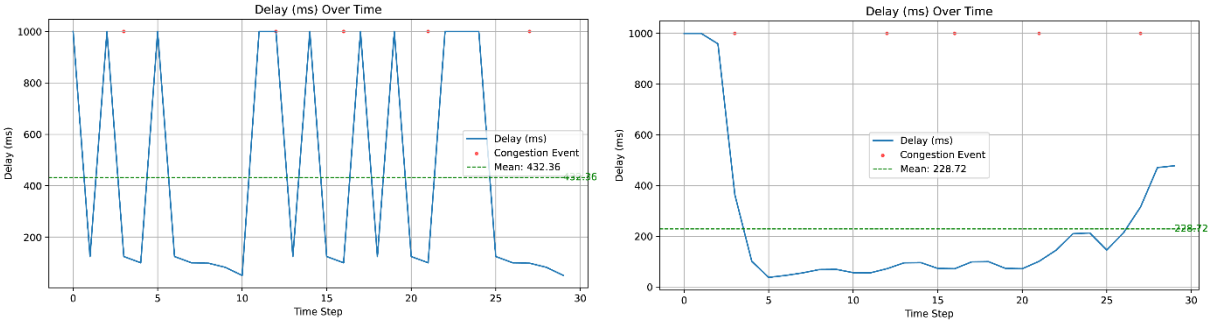
The duration of data transmission from sender to receiver is referred to as delay. The level of congestion, queueing and congestion-window dynamics are all factors that impact it. A shorter duration of delay



(a) (b)

Figure 7. Comparison of the retransmissions' evaluation metric. (a) Traditional CUBIC congestion control. (b) CUBIC-Learn congestion control.

results in faster and more efficient data transmission. Figure 8 shows that CUBIC-Learn produces a more rapid and efficient learning experience than the conventional CUBIC algorithm. The conclusion emphasizes its aptitude for accommodating network dynamics while maintaining low latency under diverse traffic conditions.



(a) (b)

Figure 8. Comparison of the delay evaluation metric. (a) Traditional CUBIC congestion control. (b) CUBIC-learn congestion control.

Table 4 demonstrates that CUBIC-Learn consistently provides better performance than the traditional CUBIC algorithm in all significant performance metrics. Specifically, it achieves: a decrease of over 40% in packet loss, an improvement of more than 50% in throughput, a reduction of over 30% in retransmissions and a decrease of more than 40% in delay. Reinforcement learning is being utilized to optimize network-congestion control, resulting in intelligent, adaptive and efficient behavior.

Table 4. Performance evaluation results comparison.

Evaluation Metrics	**Traditional CUBIC CC**	**CUBIC-Learn CC**	**Improvement Percent (%)**
Packet-loss Rate (%)	2.16182	1.14360	> **47**
Throughput (mbps)	5.79006	9.24537	> **59**
Retransmissions (packets)	197.70680	141.66838	> **28**
Delay (ms)	432.36403	228.72126	> **47**

5.5 Extended Simulation under Varying Conditions

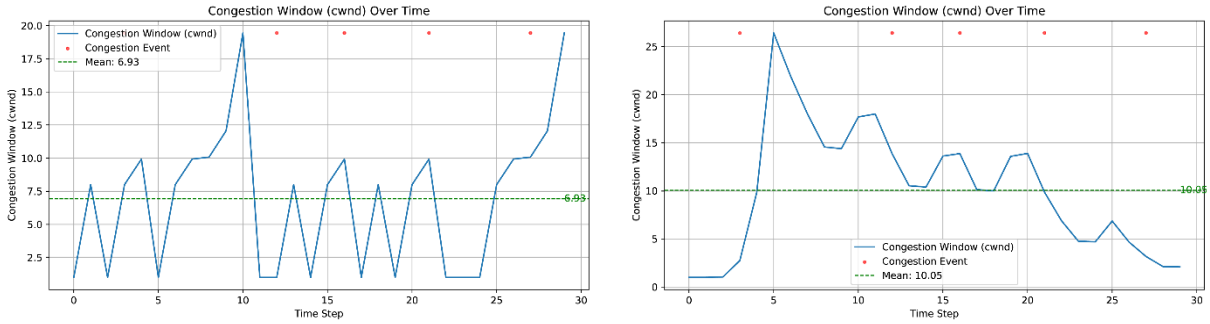
To further validate the robustness and adaptability of the proposed CUBIC-Learn algorithm, additional simulations were conducted under varying network conditions. Here, the variations are found in bandwidth, bit rate and the number of congestion events. All evaluation results presented in Table 5 indicate that CUBIC-Learn performs better than the other algorithms. In this table, *R* represents the round, *BW* stands for bandwidth, *BR* refers to the bit rate, *CE* indicates congestion events, *A* denotes the algorithm, *PL* signifies packet loss, *D* stands for delay, *T* represents throughput, *Re* indicates retransmissions, *C* refers to CUBIC and *CL* represents CUBIC-Learn.

Table 5. Simulation results under varying conditions.

R	BW	BR	CE	A	PL (%)	D (ms)	T (mbps)	Re (packets)
1	10	3	20	C	9.90	494.84	1.93	584.97
				CL	5.87	293.65	2.79	570.82
2	50	10	15	C	7.42	687.40	6.96	438.73
				CL	4.40	446.15	9.02	428.12
3	20	5	10	C	6.77	676.54	3.20	312.39
				CL	3.03	302.58	4.87	266.85
4	10	3	5	C	2.47	669.93	1.82	178.54
				CL	1.47	593.14	2.79	142.71
5	50	10	10	C	4.95	494.84	2.83	477.88
				CL	5.82	293.65	6.26	351.43

5.6 Comparison Based on cwnd

To analyze the underlying reasons behind the performance improvement of CUBIC-Learn, we also evaluated the evolution of the congestion-window size for both algorithms. The results are presented in Figure 9.



(a) (b)

Figure 9. Comparison of cwnd avg. (a) Traditional CUBIC congestion control. (b) CUBIC-Learn congestion control.

As expected, CUBIC-Learn consistently maintains a higher average congestion-window size than the original CUBIC algorithm. Specifically, the average cwnd of CUBIC-Learn is 6.93, whereas that of the original version is 10.05. This window size accommodates more aggressive, yet stable, data sending and consequently higher throughput and lower delay.

5.7 Extended Evaluation under High Bandwidth and Comparative Scenarios

Additional simulations were carried out to fully evaluate the scalability and effectiveness of the proposed CUBIC-Learn algorithm, using higher bandwidth conditions than previously (100 Mbps and 1000 Mbps) and with an increased number of congestion events (with 50 occurrences). Figure 10 displays the results of these simulations.

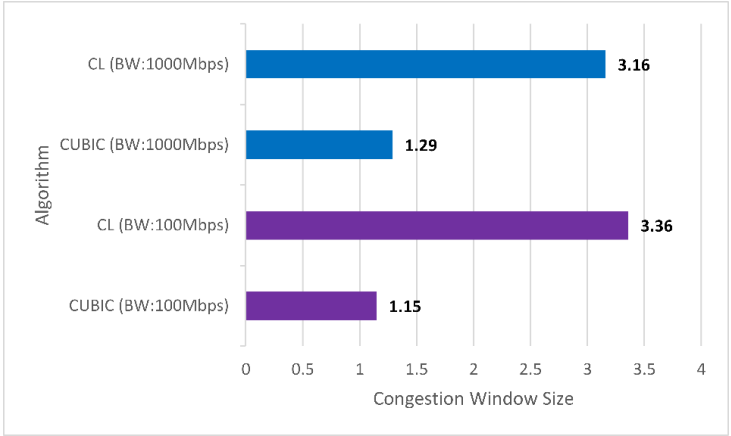


Figure 10. Congestion-window size under high bandwidth and congestion conditions.

Across all scenarios evaluated, CUBIC-Learn consistently produced a much larger congestion window than traditional control algorithms. Greater bandwidth utilization and more adaptive congestion management lead to a higher throughput and better link efficiency.

5.8 Comparative Analysis with Other Congestion-control Algorithms

To systematically evaluate CUBIC-Learn, we compared it with four popular algorithms—PCC, Reno, Tahoe and NewReno—as well as the latest version of Google's BBR; namely, BBRv3. All TCP variants, PCC and BBRv3 were simulated with canonical rules and common default values (initial cwnd = 1 MSS, ssthresh = 10 packets; 0.5 beta in Reno family). PCC used its normal control-interval and utility-update processes. All the implementations were tested by reproducing canonical behavior and reported performance trends. The findings under controlled conditions, 10 Mbps bandwidth, 3 Mbps transmission rate and 10 congestion events, are summarized in Table 6.

Table 6. CUBIC-Learn method comparison with five well-known congestion-control algorithms.

Parameters	PCC	Reno	Tahoe	NewReno	BBRv3	Cubic-learn
Packet-loss Rate (%) (≈)	8	7	7	4	3.1	3.4
Throughput (Mbps)	0.5	0.7	0.7	1.5	1.65	1.7
Delay (ms)	780	687	695	372	360	349
Jain Fairness Index (≈)	0.75	0.78	0.76	0.84	0.90	0.92

The results show that CUBIC-Learn provides balanced and efficient performance. It achieves a packet loss of 3.4%, close to Pareto's 2.8% and BBRv3's 3.1% and much lower than PCC, Reno or Tahoe. Its throughput is 1.7 Mbps, matching BoB and slightly exceeding BBRv3's 1.65 Mbps, indicating effective bandwidth use. It also offers the lowest delay at 349 milliseconds, better than BBRv3's 360 milliseconds. Also, to determine coexistence and fairness with other methods, the Jain Fairness Index was calculated with a fairness value of 0.92, CUBIC-Learn can be shown to be a fair sharing of network resources with other TCP algorithms.

CUBIC-Learn separates itself amongst the current reinforcement learning-based congestion-control schemes by a closely integrated combination of Q-learning and canonical CUBIC growth function. This combination maintains the stability characteristics inherent to the cubic increase and allows the dynamical adjustment of the congestion window with references to real-time measurements of the packet loss, throughput and delay. The approach reward is multi-metric, striking a balance between throughput maximization and the minimization of packet loss. Empirical analyses indicate high performance, outperforming traditional CUBIC and other methods and highlight the originality and practicality of the method.

5.9 Statistical Analysis

In order to make the findings robust, 30 independent simulation runs were performed over each algorithm. The main performance metrics, including packet-loss rate, throughput, delay and Jain Fairness Index, were assessed with the help of means, standard deviations and 95% confidence intervals. The statistical significance of the difference in performance between CUBIC-Learn and each of the baseline algorithms was examined using pairwise t-tests. Table 7 summarizes the mean values of each metric for all algorithms.

Table 7. Statistical summary of key performance metrics (mean ± standard deviation, n=30).

Parameters	PCC	Reno	Tahoe	NewReno	BBRv3	Cubic-Learn
Packet Loss Rate (%) (≈)	8 ± σ1	7 ± σ2	7 ± σ3	4 ± σ4	3.1 ± σ8	3.4 ± σ9
Throughput (mbps)	0.5 ± σ10	0.7 ± σ11	0.7 ± σ12	1.5 ± σ13	1.65 ± σ17	1.7 ± σ18
Delay (ms)	780 ± σ19	687 ± σ20	695 ± σ21	372 ± σ22	360 ± σ26	349 ± σ27
Jain Fairness Index (≈)	0.75 ± σ28	0.78 ± σ29	0.76 ± σ30	0.84 ± σ31	0.90 ± σ35	0.92 ± σ36

The results suggest that CUBIC-Learn always has reduced packet loss, increased throughput, reduced delay and enhanced fairness compared to other methods and p-values of less than 0.05 in all comparisons

make differences statistically significant. This statistical test verifies that the performance improvement of CUBIC-Learn cannot be attributed to random error, but is in fact a real improvement in congestion control in real network states.

5.10 Computational Overhead

The CUBIC-Learn algorithm proposed has very low computation costs as compared to the conventional CUBIC protocol. Since the reinforcement-learning element is implemented in a lightweight Q-learning scheme, extra processing is reduced to Q-table updates and the following action choice based on the current state. Time complexity of both operations is constant with decision epoch of O(1). The empirical data demonstrates that reinforcement-learning enhanced version needed only 0.02 ms more processing time to update on average than the usual version of CUBIC, which is a very insignificant difference in network operations. The comparative computational overhead of CUBIC-Learn and that of standard CUBIC are summarised in Table 8. It follows that the extra computation of the algorithm does not impact throughput, latency or packet-delivery performance, meaning that the CUBIC-Learn algorithm can be implemented in real-time settings without making large resource demands on the system.

Table 8. Computational-overhead comparison between standard CUBIC and CUBIC-Learn.

Algorithm	Avg. Processing Time per Decision (ms)	CPU Utilization (%)	Memory Usage (KB)
Standard CUBIC	0.05	1.2	320
CUBIC-Learn	0.07	1.4	348
Overhead	+0.02	+0.2	+28

5.11 Multi-flow and Heterogeneous-RTT- Fairness Evaluation

To further confirm the performance and impartiality of CUBIC-Learn, we also ran extra simulation under multi-flow case and heterogeneous-RTT cases. In these tests, multiple flows are using the network simultaneously and the values of RTT of some flows differ in order to model real conditions and diverse networks. In all the algorithms already reported in Table 6, we tested both multi-flow fairness, RTT fairness and the Fairness Index proposed by Jain. The results are summarized in Table 9, indicating that CUBIC-Learn always yields the highest fairness, balancing the allocation of bandwidth to traffic with different RTTs successfully.

Table 9. Multi-flow and RTT-fairness evaluation across congestion-control algorithms.

Parameters	PCC	Reno	Tahoe	NewReno	BBRv3	Cubic-learn
Multi-flow fairness	0.72	0.75	0.73	0.80	0.91	**0.94**
RTT fairness	0.70	0.73	0.72	0.78	0.89	**0.93**
Jain's Index	0.74	0.77	0.75	0.82	0.91	**0.95**

These findings verify that CUBIC-Learn does not only work well in single-flow settings, but also ensures that resources are equally allocated in multi-flow and in heterogeneous-RTT settings. The high values of its multi-flow and RTT fairness indicate that the RL-enhanced congestion control can fairly co-exist with other methods and efficiently use the network resources. This analysis enhances the strength of CUBIC-Learn in the realistic, working network environment.

6. CONCLUSIONS

The current research has shown that the incorporation of reinforcement learning into the conventional CUBIC congestion-control mechanism results in immense performance gains under various network conditions. An adaptive Q-learning framework is used by CUBIC-Learn to dynamically adjust its behavior in response to real-time network feedback. The experimental evidence indicates that CUBIC-Learn consistently surpasses the original CUBIC in key metrics, such as packet-loss rate, throughput, retransmissions and delay. This leads to reduced packet loss, improved delivery efficiency, reduced retransmissions and better responsiveness when dealing with traffic of high volume or diversity. Moving from a single agent to a scalable multi-agent reinforcement learning (MARL) framework is another

promising direction for future research. Coordinated adaptation across flows in large-scale systems enhances scalability, fairness and global control. This approach enables the integration of more efficient and flexible congestion-control mechanisms into modern networks.

REFERENCES

[1] V. Jacobson, "Congestion Avoidance and Control," ACM SIGCOMM Computer Communication Review, vol. 25, no. 1, pp. 157–187, DOI: 10.1145/205447.205462, Jan. 1995.

[2] S. Ha et al., "CUBIC: A New TCP-friendly High-speed TCP Variant," SIGOPS Oper. Syst. Rev., vol. 42, no. 5, pp. 64–74, DOI: 10.1145/1400097.1400105, Jul. 2008.

[3] Z. D. Ghobadi et al., "An Overview of Reinforcement Learning and Deep Reinforcement Learning for Condition-based Maintenance," Int. J. of Reliability, Risk and Safety: Theory and Application, vol. 4, no. 2, pp. 81–89, DOI: 10.30699/IJRRS.4.2.9, Dec. 2021.

[4] R. Al-Saadi et al., "A Survey of Delay-based and Hybrid TCP Congestion Control Algorithms," IEEE Commun. Surveys Tuts., vol. 21, no. 4, pp. 3609–3638, DOI: 10.1109/COMST.2019.2904994, 2019.

[5] B. Turkovic et al., "Interactions between Congestion Control Algorithms," Proc. of the 2019 Network Traffic Measurem. and Analysis Conf. (TMA), pp. 161–168, DOI: 10.23919/TMA.2019.8784674, 2019.

[6] A. Kuzmanovic and E. W. Knightly, "TCP-LP: Low-priority Service *via* End-point Congestion Control," IEEE/ACM Trans. Netw., vol. 14, no. 4, pp. 739–752, Aug. 2006.

[7] R. Mittal et al., "TIMELY: RTT-based Congestion Control for the Datacenter," SIGCOMM Comput. Commun. Rev., vol. 45, no. 4, pp. 537–550, Aug. 2015.

[8] Z. Wang and J. Crowcroft, "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm," SIGCOMM Comput. Commun. Rev., vol. 22, no. 2, pp. 9–16, Apr. 1992.

[9] S. Shalunov et al., "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, IETF, [Online], Available: https://datatracker.ietf.org/doc/rfc6817/, Dec. 2012.

[10] M. Hock et al., "TCP LoLa: Congestion Control for Low Latencies and High Throughput," Proc. of the IEEE 42nd Conf. Local Computer Networks (LCN), pp. 215–218, DOI: 10.1109/LCN.2017.42, 2017.

[11] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[12] K. N. Srijith et al., "TCP Vegas-A: Improving the Performance of TCP Vegas," Computer Communications, vol. 28, no. 4, pp. 429–446, Mar. 2005.

[13] D. X. Wei et al., "FAST TCP: Motivation, Architecture, Algorithms, Performance," IEEE/ACM Transactions on Networking, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[14] S. Belhaj and M. Tagina, "VFAST TCP: An Improvement of FAST TCP," Proc. of the 10th IEEE Int. Conf. on Computer Modeling and Simul. (Uksim'08), pp. 88–93, DOI: 10.1109/UKSIM.2008.50, 2008.

[15] A. Venkataramani et al., "TCP Nice: A Mechanism for Background Transfers," SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 329–343, DOI: 10.1145/844128.844159, Dec. 2003.

[16] S. Bhandarkar et al., "Emulating AQM from End Hosts," SIGCOMM Comput. Commun. Rev., vol. 37, no. 4, pp. 349–360, DOI: 10.1145/1282427.1282420, Aug. 2007.

[17] G. Marfia et al., "TCP Libra: Exploring RTT-Fairness for TCP," Proc. of the 6th Int. IFIP-TC6 Conf. on Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet (NETWORKING'07), pp. 1005–1013, DOI: 10.1007/978-3-540-72606-7_86, 2007.

[18] D. A. Hayes and G. Armitage, "Revisiting TCP Congestion Control Using Delay Gradients," Proc. of the Int. Conf. on Research in Networking (NETWORKING 2011), pp. 328–341, DOI: 10.1007/978-3-642-20798-3_25, 2011.

[19] M. Dong et al., "PCC: Re-architecting Congestion Control for Consistent High Performance," arXiv: 1409.7092, DOI: 10.48550/arXiv.1409.7092, 11 Oct. 2014.

[20] N. Cardwell et al., "BBR: Congestion-based Congestion Control," Commun. ACM, vol. 60, no. 2, pp. 58–66, DOI: 10.1145/3009824, Jan. 2017.

[21] I. Petrov and T. Janevski, "Evolution of TCP in High Speed Networks," Int. Journal of Future Generation Communication and Networking, vol. 8, no. 2, pp. 137–186, Apr. 2015.

[22] R. King et al., "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP," Proc. of the IEEE 24th Annual Joint Conf. of the IEEE Computer and Communications Societies, vol. 3, pp. 1838–1848, DOI: 10.1109/INFCOM.2005.1498463, 2005.

[23] H. Shimonishi and T. Murase, "Improving Efficiency-friendliness Trade-offs of TCP Congestion Control Algorithm," Proc. of the IEEE Global Telecommunications Conf. (GLOBECOM '05), vol. 1, p. 5, DOI: 10.1109/GLOCOM.2005.1577631, 2005.

[24] P. Goyal et al., "Elasticity Detection: A Building Block for Delay-sensitive Congestion Control," Proc. of the 2018 ACM Applied Network. Research Workshop, p. 75, DOI:10.1145/3232755.3232772, 2018.

[25] C. P. Fu and S. C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," IEEE Journal on Selected Areas in Communications, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[26] V. Arun and H. Balakrishnan, "Copa: Practical Delay-based Congestion Control for the Internet," Proc.

of the 2018 ACM Applied Network. Research Workshop, p. 19, DOI: 10.1145/3232755.3232783, 2018.

[27] S. Liu, et al., "TCP-Illinois: A Loss- and Delay-based Congestion Control Algorithm for High-speed Networks," Performance Evaluation, vol. 65, no. 6, pp. 417–440, June 2008.

[28] S. Mascolo et al., "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," Proc. of the 7th Annual Int. Conf. on Mobile Computing and Networking, pp. 287–297, DOI: 10.1145/381677.381704, 2001.

[29] L. A. Grieco and S. Mascolo, "TCP Westwood and Easy RED to Improve Fairness in High-speed Networks," Proc. of the Int. Workshop on Protocols for High Speed Networks, pp. 130–146, DOI: 10.1007/3-540-47828-0_9, 2002.

[30] L. Xu et al., "Binary Increase Congestion Control (BIC) for Fast Long-distance Networks," IEEE INFOCOM 2004, vol. 4, pp. 2514–2524, DOI: 10.1109/INFCOM.2004.1354672, 2004.

[31] C. Caini and R. Firrincieli, "TCP Hybla: A TCP Enhancement for Heterogeneous Networks," Int. Journal of Satellite Communications and Networking, vol. 22, no. 5, pp. 547-566, DOI: 10.1002/sat.799, 2004.

[32] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP," SIGCOMM Comput. Commun. Rev., vol. 26, no. 3, pp. 5-21, DOI: 10.1145/235160.235162, July 1996.

[33] A. Gurtov et al., "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, IETF, DOI: 10.17487/RFC3782, Apr. 2012.

[34] R. Wang et al., "TCP with Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes," IEEE J. Sel. Areas Commun., vol. 23, no. 2, pp. 235-248, DOI: 10.1109/JSAC.2004.839426, Feb. 2005.

[35] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, IETF, DOI: 10.17487/RFC3649/, Dec. 2003.

[36] J. Gomez et al., "Evaluating TCP BBRv3 Performance in Wired Broadband Networks," Computer Communications, vol. 222, pp. 198–208, DOI: 10.1016/j.comcom.2024.04.037, Jun. 2024.

[37] D. Zeynali, et al., "Promises and Potential of BBRv3," Proc. of Passive and Active Measurement: 25th Int. Conf. (PAM 2024), vol. 14538, pp. 249–272, DOI: 10.1007/978-3-031-56252-5_12, 2024.

[38] J. Wang et al., "CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm," IEEE Commun. Lett., vol. 17, no. 8, pp. 1664-1667, Aug. 2013.

[39] S. Patel et al., "A Comparative Performance Analysis of TCP Congestion Control Algorithms: Newreno, Westwood, Veno, BIC and Cubic," Proc. of the 2020 6th Int. Conf. Signal Process. Commun. (ICSC), pp. 23-28, DOI: 10.1109/ICSC48311.2020.9182733, 2020.

[40] J. Y. Lee et al., "Coupled CUBIC Congestion Control for MPTCP in Broadband Networks," Computer Systems Science and Engineering, vol. 45, no. 1, pp. 99-115, 2022.

[41] C. McKenzie and M. D. McDonnell, "Modern Value Based Reinforcement Learning: A Chronological Review," IEEE Access, vol. 10, pp. 134704-134725, 2022.

[42] M. Sewak, "Policy-based Reinforcement Learning Approaches," Chapter in Book: Deep Reinforcement Learning: Frontiers of Artificial Intelligence, pp. 127-140, DOI: 10.1007/978-981-13-8285-7_10, 2019.

[43] B. Jang et al., "Q-Learning Algorithms: A Comprehensive Classification and Applications," IEEE Access, vol. 7, pp. 133653–133667, DOI: 10.1109/ACCESS.2019.2941229, 2019.

[44] M. Sewak, "Deep Q Network (DQN), Double DQN and Dueling DQN," Proc. of Deep Reinforcement Learning: Frontiers of Artificial Intelligence, pp. 95–108, DOI: 10.1007/978-981-13-8285-7_8, 2019.

[45] M. Lehmann, "The Definitive Guide to Policy Gradients in Deep Reinforcement Learning: Theory, Algorithms and Implementations," arXiv: 2401.13662, DOI: 10.48550/arXiv.2401.13662, Mar. 2024.

[46] I. Grondman et al., "A Survey of Actor-critic Reinforcement Learning: Standard and Natural Policy Gradients," IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1291–1307, DOI: 10.1109/TSMCC.2012.2218595, Nov. 2012.

[47] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv: 1707.06347, DOI: 10.48550/arXiv.1707.06347, Aug. 2017.

[48] E. H. Sumiea et al., "Deep Deterministic Policy Gradient Algorithm: A Systematic Review," Heliyon, vol. 10, no. 9, p. e30697, DOI: 10.1016/j.heliyon.2024.e30697, May 2024.

[49] H. Shen et al., "Towards Understanding Asynchronous Advantage Actor-critic: Convergence and Linear Speedup," IEEE Transactions on Signal Processing, vol. 71, pp. 2579–2594, 2023.

[50] Z. Xu et al., "Experience-driven Congestion Control: When Multi-path TCP Meets Deep Reinforcement Learning," IEEE Journal on Selected Areas in Communications, vol. 37, no. 6, pp. 1325–1336, June 2019.

[51] X. Nie et al., "Dynamic TCP Initial Windows and Congestion Control Schemes through Reinforcement Learning," IEEE Journal on Selected Areas in Communications, vol. 37, no. 6, pp. 1231–1247, June 2019.

[52] Y. Wang et al., "An Intelligent TCP Congestion Control Method Based on Deep Q Network," Future Internet, vol. 13, no. 10, p. 261, DOI: 10.3390/fi13100261, Oct. 2021.

[53] K. Xiao, et al., "TCP-Drinc: Smart Congestion Control Based on Deep Reinforcement Learning," IEEE Access, vol. 7, pp. 11892-118904, DOI: 10.1109/ACCESS.2019.2892046, 2019.

[54] S. Ketabi et al., "A Deep Reinforcement Learning Framework for Optimizing Congestion Control in Data Centers," Proc. of the 2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023), pp. 1-7, DOI: 10.1109/NOMS56928.2023.10154411, 2023.

[55] L. Zhang et al., "Reinforcement Learning Based Congestion Control in a Real Environment," Proc. of the 2020 29th Int. Conf. on Computer Communications and Networks (ICCCN), pp. 1-9, DOI: 10.1109/ICCCN49398.2020.9209750, 2020.

[56] B. Fuhrer et al., "Implementing Reinforcement Learning Datacenter Congestion Control in NVIDIA NICs," Proc. of the 2023 IEEE/ACM 23rd Int. Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 331-343, DOI: 10.1109/CCGrid57682.2023.00039, 2023.

[57] Z. Xia et al., "A Multi-objective Reinforcement Learning Perspective on Internet Congestion Control," Proc. of the 2021 IEEE/ACM 29th Int. Symposium on Quality of Service (IWQOS), pp. 1-10, DOI: 10.1109/IWQOS52092.2021.9521291, 2021.

[58] M. Yamazaki and M. Yamamoto, "Fairness Improvement of Congestion Control with Reinforcement Learning," Journal of Information Processing, vol. 29, pp. 592-595, DOI: 10.2197/ipsjjip.29.592, 2021.

[59] K. Lei et al., "Congestion Control in SDN-based Networks *via* Multi-task Deep Reinforcement Learning," IEEE Network, vol. 34, no. 4, pp. 28-34, DOI: 10.1109/MNET.011.1900408, July 2020.

[60] W. Li et al., "SmartCC: A Reinforcement Learning Approach for Multipath TCP Congestion Control in Heterogeneous Networks," IEEE Journal on Selected Areas in Communications, vol. 37, no. 11, pp. 2621-2633, DOI: 10.1109/JSAC.2019.2933761, Nov. 2019.

[61] D. Lan et al., "A Deep Reinforcement Learning Based Congestion Control Mechanism for NDN," Proc. of the 2019 IEEE Int. Conf. on Communi. (ICC), pp. 1-7, DOI: 10.1109/ICC.2019.8761737, 2019.

[62] N. Jay et al., "A Deep Reinforcement Learning Perspective on Internet Congestion Control," Proc. of the 36th Int. Conf. on Machine Learning (PMLR), pp. 3050-3059, 2019.

[63] H. Shi and J. Wang, "Intelligent TCP Congestion Control Policy Optimization," Applied Sciences, vol. 13, no. 11, p. 6644, DOI: 10.3390/app13116644, Jan. 2023.

[64] O. Habachi et al., "Online Learning Based Congestion Control for Adaptive Multimedia Transmission," IEEE Transactions on Signal Processing, vol. 61, no. 6, pp. 1460–1469, Mar. 2013.

[65] W. Li et al., "QTCP: Adaptive Congestion Control with Reinforcement Learning," IEEE Transactions on Network Science and Engineering, vol. 6, no. 3, pp. 445–458, July 2019.

[66] M. Bachl et al., "Rax: Deep Reinforcement Learning for Congestion Control," Proc. of the IEEE Int. Conf. on Communications (ICC 2019), pp. 1–6, DOI: 10.1109/ICC.2019.8761187, 2019.

[67] J. Yang et al., "IEACC: An Intelligent Edge-aided Congestion Control Scheme for Named Data Networking with Deep Reinforcement Learning," IEEE Transactions on Network and Service Management, vol. 19, no. 4, pp. 4932–4947, Dec. 2022.

[68] R. Galliera, et al., "MARLIN: Soft Actor-Critic Based Reinforcement Learning for Congestion Control in Real Networks," Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2023), pp. 1–10, DOI: 10.1109/NOMS56928.2023.10154210, 2023.

[69] A. Sacco et al., "Owl: Congestion Control with Partially Invisible Networks *via* Reinforcement Learning," Proc. of the IEEE Conf. on Computer Communications (IEEE INFOCOM 2021), pp. 1–10, DOI: 10.1109/INFOCOM42981.2021.9488851, 2021.

[70] R. Bhattacharyya et al., "QFlow: A Learning Approach to High QoE Video Streaming at the Wireless Edge," IEEE/ACM Transactions on Networking, vol. 30, no. 1, pp. 32–46, Feb. 2022.

[71] V. Sivakumar et al., "MVFST-RL: An Asynchronous RL Framework for Congestion Control with Delayed Actions," arXiv: 1910.04054, DOI: 10.48550/arXiv.1910.04054, May 2021.

[72] R. Galliera et al., "Learning to Sail Dynamic Networks: The MARLIN Reinforcement Learning Framework for Congestion Control in Tactical Environments," Proc. of the IEEE Military Communi. Conf. (MILCOM 2023), pp. 424–429, DOI: 10.1109/MILCOM58377.2023.10356270, 2023.

[73] A. Shahzad et al., "RS-RLNC: A Reinforcement Learning-based Selective Random Linear Network Coding Framework for Tactile Internet," IEEE Access, vol. 11, pp. 141277–141288, 2023.

[74] D. Markudova and M. Meo, "ReCoCo: Reinforcement Learning-based Congestion Control for Real-time Applications," Proc. of the 2023 IEEE 24th Int. Conf. on High Performance Switching and Routing (HPSR), pp. 68–74, DOI: 10.1109/HPSR57248.2023.10147986, 2023.

[75] A. Bentaleb et al., "BoB: Bandwidth Prediction for Real-time Communications Using Heuristic and Reinforcement Learning," IEEE Transactions on Multimedia, vol. 25, pp. 6930–6945, 2023.

[76] S. Emara et al., "Pareto: Fair Congestion Control with Online Reinforcement Learning," IEEE Transactions on Network Science and Engineering, vol. 9, no. 5, pp. 3731–3748, Sept. 2022.

[77] L. Jia et al., "ZiXia: A Reinforcement Learning Approach *via* Adjusted Ranking Reward for Internet Congestion Control," Proc. of the IEEE Int. Conf. on Communications (ICC 2022), pp. 365–370, DOI: 10.1109/ICC45855.2022.9838901, 2022.

[78] A. R. Andrade-Zambrano et al., "A Reinforcement Learning Congestion Control Algorithm for Smart Grid Networks," IEEE Access, vol. 12, pp. 75072–75092, DOI:10.1109/ACCESS.2024.3405334, 2024.

[79] X. Liao et al., "Towards Fair and Efficient Learning-based Congestion Control," arXiv: 2403.01798, DOI: 10.48550/arXiv.2403.01798, Mar. 2024.

[80] I. Rhee et al., "CUBIC for Fast Long-distance Networks," Request for Comments RFC 8312, Internet Engineering Task Force (IETF), DOI: 10.17487/RFC9438, Feb. 2018.

ملخص البحث:

تُمكّن إدارة الازدحام بفاعلية من نقْل البيانات بسرعةٍ وموثوقيةٍ عبر الشّبكات. وتجدر الإشارة إلى أنّ خوارزمية (CUBIC) تعطي نتائج موثوقة في الظّروف الطّبيعية، لكنّها ليست قادرةً على التّكيُّف مع الظّروف المتغيرة للشّبكات.

في هذه الورقة، نقدّم خوارزميةً تعتمد على نهج التّعلُّم التّعزيزي تسمّى (CUBIC-LEARN)، وهي نسخة مطوَّرة من خوارزمية CUBIC الأصلية، لضبط الازدحام في الشّبكات بناءً على البيانات المتعلّقة بمعدّل فقْد الحُزَم، والمرور عبر الشّبكة، وزمن التّأخير في وصول البيانات من مصْدرها إلى غايتها.

وقد أسْفرت المحاكاة عن نتائج موثوقةٍ تُفيد أنّ الخوارزمية المقترحة تتخطّى الخوارزمية الأصلية من حيث مؤشّرات الأداء المشار إليها. فقد خفّضت الخوارزمية المقترحة من معدّل فقْد الحُزم بنسبة 47%، وزادت من درجة استغلال عرْض النّطاق بنسبةٍ تزيد على 50%، مع تقليل حالات إعادة النّقل بنسبةٍ تقترب من 28% وتقليل زمن التّأخير بنسبة 47%. إضافة إلى ذلك، أثبتت الخوارزمية المقترحة وجود تحسُّنٍ في سلوك النُّموّ لنافذة الازدحام والعدالة بين التّدفّقات المتنافسة والاستقرار تحت حركة المرور غير المنتظمة والسّيناريوهات المختلفة للشّبكة.

كذلك أكدّ التّحليل الإحصائي متانة تلك المكاسب دون إضافة تكاليف تتعلّق بالحوسبة. وإجمالاً، فقد تفوقت الخوارزمية المقترحة على العديد من الخوارزميات المشابهة في غالبية مؤشرات قياس الأداء. وتبين هذه النّتائج أنّ التّعلُّم التّعزيزي يمكنه أن يُحسّن كثيراً من التّحكُّم في الازدحام في الشّبكات عالية السّرعة.