TCP-PPCC: Online-Learning Proximal Policy for Congestion Control

Jing Li, Yuyao Guan, Pengpeng Ding, Shiwei Wang
Shanghai University of Electric Power
Shanghai, China
lijing@shiep.edu.cn, {cherylguan, shylock, wangshiwei}@mail.shiep.edu.cn

***Abstract*—Effective network congestion control strategies are the key to secure the normal operation of complex and changeable networks. The fundamental assumptions of many existing TCP congestion control variants dominated by hand-crafted heuristic algorithms are no longer valid. We propose an algorithm called TCP-Proximal Policy Congestion Control (*TCP-PPCC*), which is based on deep reinforcement learning algorithm Proximal Policy Optimization (PPO). TCP-PPCC updates the policy offline from the features of the preceding network state and feedback from the current network environment and adjusts the congestion window online with the updated policy. The senders with TCP-PPCC can learn about the changes in network bandwidth more accurately and adjust the congestion window in time. We demonstrate the performance of TCP-PPCC by comparing it with the traditional congestion control algorithm NewReno in four network scenarios with the ns-3 simulator. The results show that in scenario 2, TCP-PPCC takes 58.75% improvement in average delay and 27.80% improvement in throughput compared with NewReno.**

***Index Terms*—TCP congestion control, Deep Reinforcement Learning, Proximal Policy Optimization, ns-3 simulator**

I. Introduction

Nowadays, network traffic has developed unprecedentedly. TCP protocol providing connection-oriented and reliable data transmission services, congestion control (CC) is an important proposition in TCP protocol. CC is intended to manage network resources efficiently and to provide resource sharing among competing flows while protecting the network from collapse.

Although the artificial heuristic algorithms take wonderful effects in the early simple network, the traditional CC algorithms can no longer explore the available bandwidth and take full advantage of network resources in the complex network environment of recent years. Many researchers try to use machine learning methods to solve CC problems. Two principal schemes were proposed: (1) Remy, training offline in a simulator [1]; (2) PCC, training online but discarding the experience it has obtained [2].

To address the limitations of the above algorithms, congestion control schemes based on reinforcement learning (RL) has been proposed recently [3–5]. In this paper, we propose an improved algorithm based on the deep reinforcement learning (DRL) algorithm for CC. The main contributions of this paper are summarized as follows:

This work was supported by the National Natural Science Foundation of China (No. 61872230, No. 61572311).

1) We regard CC as a Markov decision process and design special RL elements including observation space, action space, and reward function for it. The training framework we use is Proximal Policy Optimization.
2) We propose an algorithm called TCP-PPCC, which consists of two parts: (i) periodic training the DRL model offline to update and optimize the parameters of the policy; (ii) determining the value of the *cwnd* online with the current network state and the policy trained offline. It is both an experienced and exploratory DRL agent.
3) We implement this DRL framework based on Keras and the ns3-gym interface and evaluate the performance of it in four scenarios with the ns-3 simulator. The result is shown that TCP-PPCC provides a good tradeoff between throughput and delay compared with NewReno and QTCP.

II. Related work

A. Rule-based protocol

According to the signal in judging the congestion of the network, the algorithm can be divided into loss-based and delay-based. A series of mechanisms which have become the cornerstone of many congestion algorithm schemes were introduced in TCP Reno [6]. Sangtae et al. [7] proposed a cubic function for the *cwnd* modification. In these schemes, the packet loss event is recognized as a link congestion signal.

To solve the problem of over-occupation with the buffer by loss-based algorithms, they use the delay as a congestion signal [8, 9]. If the delay exceeds a certain threshold, the *cwnd* is reduced to relieve congestion. While the link is congested, the delay-based algorithm will respond earlier than the loss-based algorithm.

B. Intelligent schemes

These algorithms described above are built on some fundamental assumptions. When the network environment develops complex, it will be tough for them to solve it. Recently, some TCP CC schemes based on machine learning methods were proposed. Remy [1] adopted machine learning to the sending rate decision of congestion control for the first time. However, it depends on the formation of training data. The performance will be significantly reduced when the actual network conditions changing. PCC [2] adjusts the sending rate adaptively by training online but discarding the experience obtained.

QTCP [3] is a TCP CC scheme based on Q-learning. It approximates the Q-function with Kanerva Coding. Kong et al. [4] designed two learning-based TCP CC schemes: LP-TCP and RL-TCP. Jay et al. [5] proposed PCC-RL, which is a RL congestion control agent with the test environment based on the OpenAI Gym interface.

III. DEEP REINFORCEMENT LEARNING FOR CONGESTION CONTROL

Reinforcement learning solves sequential decision-making problems. The typical reinforcement learning algorithm includes two entities: *agent* and *environment*. The environment embraces observation space, action space, and reward function.

A. Environment

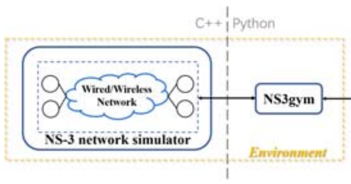


Fig. 1. The architecture of the environment of DRL for CC. It consists of two parts: the ns-3 network simulator and the ns3-gym [10] interface.

In this work, the architecture of the environment, shown in Figure. 1. As a RL problem, TCP-PPCC consists of the following elements:

1) Observation Space: According to the characteristics of network congestion, in a TCP connection, we consider the features of the state s_t listed in Table. I.

TABLE I
THE STATE s_t OF OBSERVATION SPACE.

	Feature	Description
1	ssThresh	Current slow start threshold
2	cwnd	Size of congestion window sizes at time t
3	segment	Size of segment at time t
4	inflight_sum	Number of unacked packets before time t
5	inflight_avg	Mean number of unacked packets at time t
6	acked_sum	Number of acked packets before time t
7	acked_avg	Mean number of acked packets at time t
8	rtt_avg	Mean RTT at time t
9	rtt_min	Minimum RTT in network environment
10	tx_avg	Mean sent interval of two acked packets
11	rx_avg	Mean received interval of two acked packets
12	throughput	Throughput measurement at time t

Then we combine M states of continuous past time intervals into a system state, denoted as a two-dimension tensor, which is described as $S = [s_0, s_1, \dots, s_{M-1}]$.

2) Action Space: Within the congestion control problem, the actions here are to adjust the size of the *cwnd*. To limit the fluctuation of window value from affecting the stability of the network too much, we design action as:

$$cwnd_{t+1} = cwnd_t + a_t \cdot \frac{MSS^2}{cwnd_t}, \tag{1}$$

where $a_t \in A$ = {-200, -100, -50, -20, 0, 5, 10, 100, 200, 500, 600}, and the MSS is the maximum data segment size of each TCP packet transmission. We promote the agent to increase the *cwnd* during training. Therefore, we create the action value to be larger and more positive actions to increase *cwnd* than to reduce them.

3) Reward Function: The setting of Rewards determines the direction of the entire training process. In this work, we first design a utility function U as:

$$U = \alpha \times \log(thoughput) - \beta \times \log(rtt), \tag{2}$$

where $\alpha + \beta = 1$. The value of reward r_t at time t is determined by observing the difference of two consecutive utility values $\Delta U = U_{t+1} - U_t$. Then we define r_t as:

$$r_t = \begin{cases} 1, & \Delta U > 0 \\ 0, & \Delta U = 0. \\ -1, & \Delta U < 0 \end{cases} \tag{3}$$

B. Agent

In this work, we design the agent with an actor-critic DRL framework, which can learn from whole rewards and TD errors. The following two parts are presented for Actor and Critic networks respectively.

1) Critic Network: This neural network is a "critic" based on a value function, when the actor neural network chooses an action a_t and performs it, the state of the environment changes. The critic neural network takes the state s_t as input and gets the state-value function from the transformed state of the environment and the reward:

$$\begin{aligned} V^\pi(s) &= E_\pi\left[\sum_{t=0}^{\infty} \gamma \cdot r_t\left(s_t, a_t | s_0 = s\right)\right] \\ &= E_\pi[r_t(s_t, a_t) + \gamma \cdot V^\pi(s_{t+1}) | s_0 = s]. \end{aligned} \tag{4}$$

Then, to achieve the advantage function (i.e., the TD error):

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s), \tag{5}$$

the critic neural network returns the state-action value as follow to the update of the actor neural network:

$$Q^\pi(s, a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma \cdot r_t(s_t, a_t) | s_0 = s, a_0 = a\right], \tag{6}$$

where γ is the discount factor.

2) Actor Network: The actor-network is updated using Proximal Policy Optimization (PPO) [11]. It can limit the update range of the new strategy by the ratio of the new strategy of the old strategy and set the confidence interval of the gradient update to ensure the stability. The loss function of the actor-network is:

$$L(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right], \tag{7}$$

where π_θ is a random policy function, β is the penalty coefficient, $\hat{A}_t$ is an estimate of the advantage function at time t, which is given by the Critic neural network.

IV. DRL TRAINING CONFIGURATION AND ARCHITECTURE

A. Testbed Environment

In this work, we consider a classical dumbbell-shaped network topology as an environment indicated in Figure. 2. We deploy our algorithm at the sender, which is the agent, to learn the changes of network state online and adjust the congestion window in real-time.

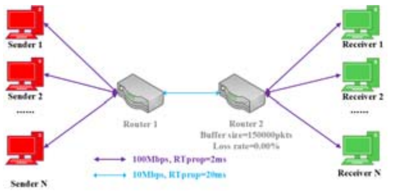


Fig. 2. The dumbbell-shaped network topology with N senders, N destinations, and two routers simulation scenarios implemented in the ns-3 simulator. We set the bottleneck bandwidth as B=10 Mbps and RTT_{min}= 48 ms by default.

To test the adaptability of the algorithm to the network environment better, we set up four scenarios, as shown in Table. II.

TABLE II
SCENARIOS FOR TESTING NETWORK ADAPTABILITY.

Scenarios	The buffer size of two Routers	The loss ratio
1	150000 packets	0.00%
2	150000 packets	0.01%
3	150 packets	0.00%
4	150 packets	0.01%

B. Agent Network Model Training

We utilize Keras to implement the design of the agent. It trains in the way of the Actor-critic algorithm. The framework of TCP-PPCC is illustrated as Figure. 3.

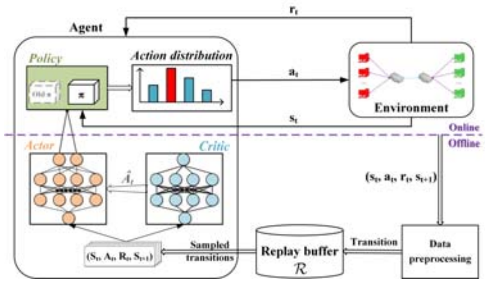


Fig. 3. The framework of TCP-PPCC. It consists of the online decision-making process and the offline training process.

1) Online Decision-making: In the online decision-making process, the agent performs an action a_t got from the action probability distribution so the state of the environment s_t changes to s_{t+1}, and it returns a reward r_t to the agent.

2) Offline Training: In the offline training process, we collect the data from the interactions between the agent and environment. Due to the network state data has different dimensions (e.g., throughput and rtt_min). In this work, we apply linear normalization to map the input data to the range of $[0, 1]$ depending on:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{8}$$

After the data preprocessing, we store each transition (s_t, a_t, r_t, s_{t+1}) into the replay buffer $\mathcal{R}$. Then the trainer uses the data sampled in replay buffer to train the actor and the critic neural network model.

V. EVALUATION

In this section, we evaluate the performance of the TCP-PPCC we propose in the ns-3 simulator. TCP CC schemes for comparing the performance are NewReno [12], QTCP[1] [3], and the TCP-PPCC we propose.

The metrics we utilize are the throughput thr of a TCP flow, the mean delay t_d of a TCP flow ($t_d = RTT - RTT_{min}$), and the packets loss rate l ($l = \frac{total\ packets\ lost}{total\ packets\ sent}$). In addition to the usual metrics mentioned above, we also use a throughput-delay tradeoff metric *Mt* ($Mt = log(E(t_d)) - log(E(thr))$). The target of CC is high throughput and low latency, so the smaller *Mt*, the better network performance.

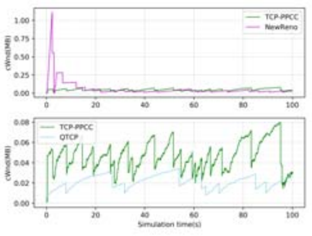
A. Experiment I: One sender.

In this experiment, we take N=1. We test the performance of them under four scenarios described in section. IV-A, and the results are shown in Table. III. It shows QTCP and NewReno cannot trade off the throughput and delay very well. On the other hand, the TCP-PPCC we propose can well balance the relationship between throughput and delay, it is demonstrated by the smaller value of *Mt* than QTCP and NewReno in most scenarios. The packet loss rate is the lowest of the three algorithms. Figure. 4 shows the network performance of NewReno, QTCP, and TCP-PPCC simulated for 100 seconds in scenario 2.

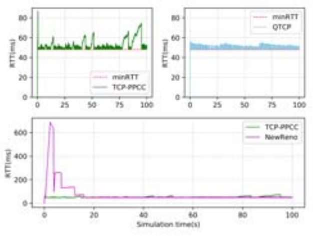
TABLE III
AVERAGE PERFORMANCE OF A SINGLE TCP SENDER WITH NEWRENO, QTCP, AND TCP-PPCC.

Scenarios	Algorithm	$E(t_d)$	$E(thr)$	l	Mt
1	NewReno	1862.83	**6.83**	0	2.44
	QTCP	**34.74**	5.92	0	**0.77**
	TCP-PPCC	46.04	6.37	0	0.85
2	NewReno	68.44	4.46	2.89184E-05	1.19
	QTCP	**24.57**	2.78	2.95069E-05	0.95
	TCP-PPCC	28.23	**5.70**	**2.72537E-05**	**0.69**
3	NewReno	65.85	**6.86**	0	0.98
	QTCP	**24.54**	3.23	0	0.88
	TCP-PPCC	47.38	6.73	0	**0.84**
4	NewReno	25.19	4.39	2.90522E-05	0.76
	QTCP	**24.56**	2.04	3.07017E-05	1.08
	TCP-PPCC	26.66	**5.02**	**2.82222E-05**	**0.72**

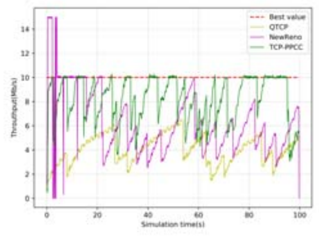
[1]Since it has no exposed code, we implement QTCP to the best of our ability according to [3].



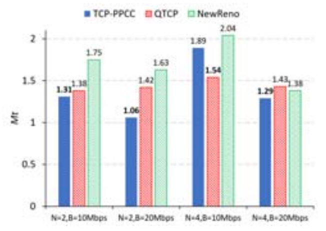
(a) Real-time *cwnd*.



(b) Real-time RTT.



(c) Real-time throughput.

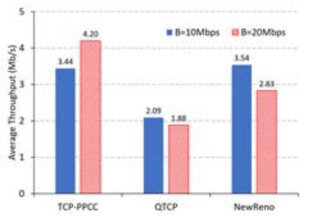


(d) The *Mt* of multiple senders N.

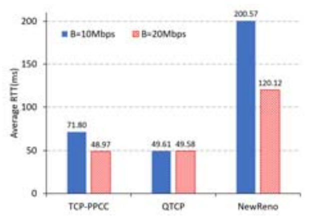
Fig. 4. Comparisons of simulation results of *cwnd*, RTT and throughput of NewReno, QTCP and TCP-PPCC in scenario 2. (a) shown, TCP-PPCC keeps the *cwnd* within an appropriate range. In (b) and (c), TCP-PPCC realizes higher performance in RTT and throughput than NewReno and QTCP within most of the time during the simulation. (d) The *Mt* of multiple senders N over NewReno, QTCP, and TCP-PPCC algorithms under fluctuant bottleneck bandwidths B.

B. *Experiment II: Multiple senders.*

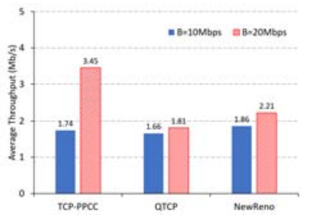
In this experiment, we evaluate the average RTT and the average throughput per receiver when multiple senders adopting the same CC algorithm with the bottleneck bandwidth B=10 Mbps and B=20 Mbps. We set the number of senders as N=2 and N=4 respectively. Similarly, we also simulate 100 seconds under scenario 2. The results are shown in Figure. 5.



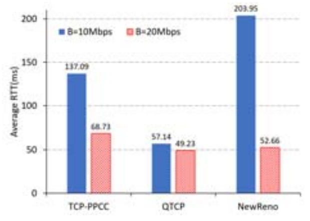
(a) Throughput of two senders.



(b) RTT of two senders.



(c) Throughput of four senders.



(d) RTT of four senders.

Fig. 5. Average RTT and throughput of multiple senders over NewReno, QTCP, and TCP-PPCC algorithms under fluctuant bottleneck bandwidths B.

The average throughput per receiver with TCP-PPCC is better adapted to bandwidth increasing than that with QTCP and NewReno. From the compromise metric, the *Mt* exhibited in Figure. 4(d), we can recognize that TCP-PPCC balances RTT and throughput better than the other two algorithms in most scenarios of the network configurations we design.

VI. CONCLUSION

In this paper, we propose an intelligent congestion control algorithm called TCP-PPCC. TCP-PPCC determines the value of the congestion window online with the models which are trained offline with the data collected from the ns-3 simulator. It uses the ratio between the new policy and the old policy to limit the update range of the new policy and sets a confidence interval for the gradient update to ensure stability. By adjusting the setting of the element reward of reinforcement learning, the influence of the delay in the continuous interaction between the environment and the agent on the congestion control problem is weakened. The results obtained with the ns-3 simulator show that TCP-PPCC provides a good tradeoff between throughput and delay compared with NewReno and QTCP.

References

[1] 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.

[2] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, “Pcc: Re-architecting congestion control for consistent high performance,” *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 395–408, 2015.

[3] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, “Qtcp: Adaptive congestion control with reinforcement learning,” *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2018.

[4] Y. Kong, H. Zang, and X. Ma, “Improving tcp congestion control with machine intelligence,” *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pp. 60–66, 2018.

[5] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, “A deep reinforcement learning perspective on internet congestion control,” *International Conference on Machine Learning*, pp. 3050–3059, 2019.

[6] V. Jacobson, “Congestion avoidance and control,” *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.

[7] 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.

[8] 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, 1995.

[9] C. Jin, D. X. Wei, and S. H. Low, “Fast tcp: motivation, architecture, algorithms, performance,” *IEEE INFOCOM 2004*, vol. 4, pp. 2490–2501, 2004.

[10] P. Gawłowicz and A. Zubow, “ns-3 meets openai gym: The playground for machine learning in networking research,” *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 113–120, 2019.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[12] S. Floyd, T. Henderson, and A. Gurtov, “Rfc3782: The newreno modification to tcp’s fast recovery algorithm,” 2004.