A TCP Congestion Control Algorithm Based on Deep Reinforcement Learning Combined with Probe Bandwidth Mechanism

Mengting Li
School of Software, Yunnan University, Kunming, China
jkc265616@mail.ynu.edu.cn

Xiang Huang
School of Software, Yunnan University, Kunming, China
2580067062@qq.com

Chenyang Jin
School of Software, Yunnan University, Kunming, China
3216380542@qq.com

Yijian Pei*
School of Information Science & Engineering, Yunnan University, Kunming, China
yndxpyj@163.com

ABSTRACT

The rapid development of emerging Internet services such as live video, 5G, VR, and the Internet of Things puts forward higher requirements for network throughput, Latency, jitter, and loss. However, the inefficient bandwidth utilization rate of the existing TCP protocol cannot meet these requirements. Based on this problem, this paper proposes an algorithm RL-explore that uses RL (Reinforcement learning) combined with bandwidth detection mechanism. The model trained with this algorithm can effectively use the network bandwidth, and compared to other RL algorithms, it is easier to converge during training.

CCS CONCEPTS

• **Networks**; • **Network protocols**; • **Transport protocols**; • **Computing methodologies**; • **Artificial intelligence**; • **Control methods**;

KEYWORDS

Congestion control, Deep reinforcement learning, Probe Bandwidth mechanism

ACM Reference Format:
Mengting Li, Xiang Huang, Chenyang Jin, and Yijian Pei*. 2021. A TCP Congestion Control Algorithm Based on Deep Reinforcement Learning Combined with Probe Bandwidth Mechanism. In *The 5th International Conference on Computer Science and Application Engineering (CSAE 2021), October 19–21, 2021, Sanya, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3487075.3487119

1 INTRODUCTION

As the core of TCP, congestion control directly affects the transmission performance of TCP traditional congestion control algorithms can obtain feedback information of network condition by sending packets. Based on the feedback information, the congestion control algorithm will adjust the congestion control window or data transmission rate according to the predefined actions, so as to avoid network congestion. The main goal of congestion control algorithm is to ensure high bandwidth utilization and low delay while pursuing fairness and stability. At present, CUBIC [1] and BBR [2] algorithms are still widely used in the Internet.

CSAE 2021, October 19–21, 2021, Sanya, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8985-3/21/10…$15.00
https://doi.org/10.1145/3487075.3487119

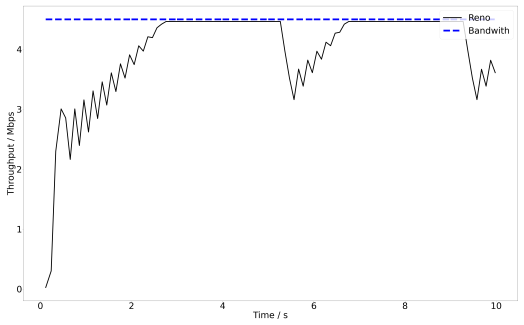

Figure 1: Timeline Diagram of TCP Reno Algorithm.

1.1 Loss-Based Congestion Control

Loss-based is to assume that loss is network congestion as a prerequisite, and adopt a slow detection mechanism to gradually increase cwnd. When packet loss occurs, cwnd and ssthresh will be reduced, such as Reno [3], Cubic, etc. In TCP Reno, the process of congestion control is divided into four stages: (1) Slow start stage. When there is no loss, every time Sender receives an ACK, CWND adds a MSS (Maximum Segment Size). At this stage, CWND doubles every round. If packet loss occurs, CWND is halved and enters (2) congestion avoidance phase: the window adds one MSS every round, which is a linear increase. When three repeated ACKs of the same message are received, it is considered that the next message of the message is lost. Enter (3) fast retransmission phase, do not wait for timeout retransmission, but immediately retransmit the lost message; after completion, enter (4) fast recovery phase, ssthresh is modified to half of the current CWND, and CWND is equal to ssthresh. Enter (1) the congestion avoidance phase and repeat the above process.

Figure 1 shows the corresponding throughput changes of TCP Reno in four stages. In the initial slow start phase within a few tenths of a second, the throughput exponentially rises. After encountering loss, it enters the congestion avoidance stage, and the throughput increases linearly. After the throughput has been maintained near the rated bandwidth for a period of time, congestion occurs and enters the fast retransmission phase.

The disadvantage of Loss-Based CC is that the throughput of the algorithm drops rapidly whenever there is a loss. Even in a network without random loss like Figure 1, its bandwidth utilization is not high. In a network with large random loss, the behavior of halving the window size will seriously reduce its performance. This bad situation not only occurs on a single link, but also on multiple links. The low channel utilization of the MPTCP protocol stems from the abrupt cwnd growth policy [4].

1.2 RL-Based Congestion Control

The characteristic of this kind of algorithm is that there is no specific congestion signal, but with the help of deep neural network, based on the training data, using deep reinforcement learning method to train a control strategy. RL-based abandoned the traditional decision mechanism of congestion control, and let the decision network of DRL to choose the adjustment action of congestion control. The purpose of these efforts is to improve the overall performance of TCP data transmission in traditional Internet architectures, such as high throughput and low latency, and to ensure a certain degree of fairness and robustness as far as possible. RL-based tries to shield the difference of the underlying network environment and uses a general congestion control algorithm model to deal with different network environment. This approach is more dependent on the input training set (historical network model), as well as how to select the state, action, reward and training algorithm of reinforcement learning.

1.3 ProbeBW Combined with RL-Based

In the process of training using the RL algorithm, we found that it takes a long time for the model to finally converge. So is there any way to make the model converge faster? Our proposed RL algorithm combined with ProbeBW (Probe Bandwidth) mechanism can do this. Simply put, we have added a constraint mechanism to DRL. We use the ProbeBW mechanism. Under the premise that the RL policy does not choose to reduce the send rate, the policy will periodically explore environment bandwidth. This ensures that we can obtain the true bandwidth of the network environment in most cases. Next, enter the detected real bandwidth as part of the state into the RL policy. Obviously, this helps RL save the process of learning to detect network bandwidth. Therefore, it is not difficult to understand that the convergence rate of RL-explore exceeds RL. Figure 2 shows that the cumulative discount reward of RL-explore also exceeds RL.

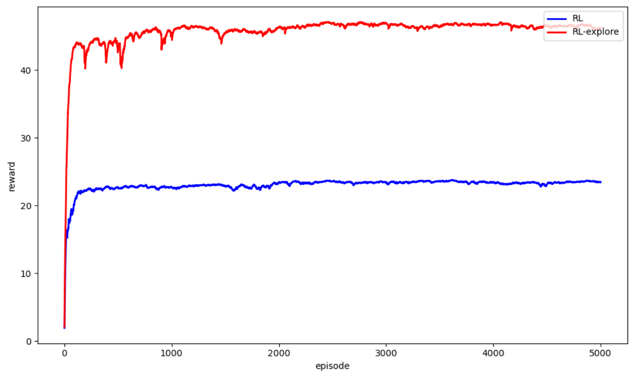

Figure 2: Convergence Speed Difference between RL-explore (Our Approach) and RL-Based.

2 RELATED WORK

As a reliable transport layer protocol, the TCP protocol ensures its reliability through a series of mechanisms such as packet loss retransmission, flow control, and congestion control.

Most TCP protocols are optimized for congestion control. Most TCP protocols achieve the purpose of congestion control by controlling the change of CWND. If there is congestion, CWND will be reduced; otherwise, CWND will be increased (such as Reno and Cubic). The traditional work of congestion control is mostly based on the simplified network model (mainly packet loss) to judge the network state, and adopts the heuristic algorithm to adjust the CWND. In addition, some researchers abandon the use of packet loss to determine congestion, and instead choose to detect network Bandwidth and RTT (such as BBR) to avoid the Bandwidth drop caused by random packet loss. With the development of machine learning technology and the emergence of Nature DQN [5], more and more researchers try to apply machine learning technology to TCP congestion control.

2.1 Performance-Oriented Congestion Control

Like all machine learning researchers, they abandon the traditional congestion control mechanism that uses heuristic algorithms to make decisions, and instead let the trained DRL (Deep Reinforcement learning) decision-making network choose the congestion control adjustment actions. The purpose of these works is to ensure a certain fairness and robustness on the basis of improving the overall performance (such as throughput and low delay) of TCP data transmission. These works focus on how to choose the states, actions, reward functions and training algorithms of reinforcement learning.

Li et al. [6] proposed QTCP, an adaptive congestion control algorithm based on Q-learning, in 2018. Its utility function is:

$$U = a \cdot \log(throughput) - b \cdot \log(RTT) \tag{1}$$

The states are selected as average time between packets sent, average RTT, and average time between packets receiving ACK. In the action space, discrete actions are selected, which are increased by 10Byte, decreased by 1Byte and kept the same. QTCP performs better than TCP New Reno [7] in many network scenarios.

Yan et al. [8] proposed Indigo, a congestion control algorithm based on imitation learning, in 2018. Indigo uses the repeatable experiment simulator Pantheon for offline training, so it can acquire network scenes as expert knowledge, thereby using imitation learning as its training algorithm. Indigo's policy network is a single-layer LSTM network, and the state space uses continuous states, such as the weighted moving average of queuing delay, sending rate, receiving rate, etc., the size of the current CWND, and the actions taken in the previous step. The action space is divided into 5 discrete actions, which are carried out to CWND respectively "/2", "-10", "+0", "+10", "×2" adjustment operation. Indigo shows superior performance in trained scenarios. Unfortunately, due to the need to carefully select expertise for each training scenario, Indigo can only train in a limited number of network scenarios, which means its generalization ability can't be guaranteed.

Jay N et al. [9] proposed Aurora in 2019. Aurora uses the delay gradient, delay ratio, and send ratio as input states. At the same time, Aurora sets the input state to the state combination of 10 time slices before the current moment, so as to obtain more sufficient historical information. Aurora's action space uses the following formula:

$$x_t = \begin{cases} x_{t-1}\left(1+\alpha a_t\right), & a_t \geq 0 \\ \frac{x_{t-1}}{1+\alpha a_t}, & a_t < 0 \end{cases} \tag{2}$$

Where x_t is the sending rate of the data packet, a_t is output by the neural network, and a is a constant. Aurora has shown more performance than traditional algorithms in both fixed-bandwidth and dynamic-bandwidth single-link network scenarios.

2.2 Congestion control for specific transmission problems

Nie et al. [10] proposed TCP-RL in 2019 to dynamically select the initial congestion window. When they measured the Baidu mobile search service, they found that more than 80% of the TCP data stream was transmitted during the slow start phase of the algorithm. For these data streams, the key element that determines their bandwidth utilization is not the adjustment mechanism of the congestion control rate, but the initial congestion window. Based on this, TCP-RL regards the initial congestion window selection as a multi-armed slot machine problem and uses the discounted UCB algorithm to train it. In the rate adjustment stage, TCP directly uses the output of the neural network to select one of the 14 existing congestion control algorithms as the adjustment algorithm for the current time period. Researchers deployed TCP-RL in a data center of Baidu's mobile search service. After a year of measurement, TCP-RL can improve the average TCP response time by 23%.

2.3 Congestion Control Combined with Traditional Algorithms

The pure data-driven DRL congestion control algorithm has the problems of low stability and low adaptability. The traditional congestion control algorithm has proved its stability after years of actual deployment and operation. Recent research work has begun to incorporate the domain knowledge of traditional congestion control algorithms (Cubic, BBR) into the design of DRL congestion control algorithms. For example, use traditional congestion control algorithms to accelerate the training of DRL agents, or use DRL to directly improve and optimize the decision-making mechanism of congestion control algorithms.

DeepCC [11] proposed by Abbasloo et al. can optimize the Cubic algorithm. As we all know, Cubic algorithms tend to fill up the buffer, so their adjusted cwnd is usually greater than the optimal. DeepCC combined with Cubic's adjustment results to establish a maximum value to avoid sending in this situation. The simulation results show that DeepCC improves the average RTT of traditional algorithms without sacrificing bandwidth utilization.

The Eagle [12] proposed by Emara et al. in 2020 utilizes the expert knowledge of BBR algorithm to optimize the trained DRL congestion control algorithm. Eagle designs BBR-like exponential startup, empties queues and detects bandwidth in three different state phases. Secondly, Eagle uses a synchronous BBR algorithm to speed up the training process and optimize the movement selection strategy by periodically generating some adjustment actions into the training. Unfortunately, the experimental results show that Eagle performs better than other classical algorithms but not significantly better than BBR.

Abbasloo et al. proposed a hybrid algorithm based on Cubic and DRL [13]. By testing the performance of DRL-based algorithms and TCP Cubic on dynamic links, they found that DRL-based algorithms have problems such as slow convergence speed and higher computational complexity. Therefore, in the hybrid algorithm, DRL is responsible for collecting information in each fixed time period and calculating a basic CWND. Then in the next period, Cubic adjusts this CWND according to its own control logic. Experiments show that in this hybrid algorithm, DRL can help Cubic adapt to changes in network bandwidth faster.

3 DRL APPROACH IS TO CONGESTION CONTROL

This section gives a brief introduction to RL, DRL, and why DRL can be used in congestion control.

3.1 Background: RL and DRL

In RL [14] (reinforcement learning), the task of the agent is to solve the problem of sequential decision-making, which is realized through the interaction with the Environment. This problem can be modeled by the Markov Decision Process (MDP). Simply put, MDP is a cyclical process in which an Agent interacts with the Environment by taking Action to change its State and obtain Reward. MDP can be described by the five-tuple $\langle S, A, P, R, \gamma \rangle$, where A represents the action space, P is the state transition probability, and $P(s'|s,a)$ represents the probability of transitioning to state s' after taking action a in state s, R is the reward function, and $\gamma \in [0,1]$ is the discount factor, which reflects the importance of the agent to the future reward. The strategy of the reinforcement learning agent is represented by $\pi : S \times A \longrightarrow [0,1]$, and the agent chooses actions according to the strategy π.

Traditional reinforcement learning uses a tabular approach to represent strategy π. However, when the state-action space is large, the table method cannot continue to be used, because the storage of tables consumes a very large amount of memory, which is also the bottleneck that prevents traditional reinforcement learning from being widely used.

In DRL [15] (Deep Reinforcement Learning), we use neural network with parameter θ to approximate strategy π, and use π_θ to represent strategy.

3.2 RL Applied to Congestion Control

The flow of congestion control can be summarized as follows: infer the network state according to the network feedback information, so as to adjust the CWND or send rate.

Traditional heuristic congestion control algorithms are often based on ideal assumptions about the network state, and use predefined actions to adjust CWND according to different network states, such as reducing CWND when packet loss occurs.

The RL-Based congestion control process can be abstracted as a partially observable Markov decision process. States are network statistics collected in the past period of time. After the sender has determined the sending rate x_t, it will observe the results sent at that rate and determine the statistical vector based on the received data packets.

Action will change the sending rate. In this article, the Agent's Action is to change the sending rate.

Reward settings. Reward is usually related to the delay, packet loss rate, and sending rate of network statistics. The weight setting of specific parameters is related to specific applications. For example, the weight of the sending rate can be increased for large file transmission applications, and the weight of delay can be increased for online games that require low latency.

4 MODEL DESIGN

Aiming at the adaptability of traditional congestion control algorithms, we propose and implement the Probe Bandwidth mechanism combined with the adaptive congestion control algorithm RL-explore of the deep reinforcement learning framework. This chapter mainly introduces the background knowledge of deep reinforcement learning, the algorithm design and specific implementation of RL-explore. It should be noted that the work in this chapter only discusses the adaptability of TCP flows under a single link.

4.1 Architecture

We model the congestion control problem as a sequential decision problem under the framework of reinforcement learning. The specific meanings of agents, states, actions, and rewards in the reinforcement learning framework are as follows:

Agent. Set on the sender of the TCP connection. Its main job is to observe the network status and adjust the sending rate according to the decision-making network. The decision-making network is a fully connected layer network consisting of three layers.

State. It is composed of data collected in one or more statistical intervals in the past. Each time a packet is received, the event time and event type (ACK or DROP) of the current packet will be saved. After each statistical interval is over, the throughput, latency, loss and other information in the last period of time can be obtained by calculation. The Latency is the average latency of each packet. The state of RL-explore is composed of $throughput_{max}$, *loss*, *send rate*, *avg_latency*. Of course, these data have been normalized before being used. In the t-th statistical interval, the observed state can be expressed as:

$$obs = (throughput_{max},\ loss, send\ rate, avg\_latency) \qquad (3)$$

Where $throughput_{max}$is the maximum throughput in the past 5 time periods, and *loss* is the packet loss rate in the last time period, the *send rate* is the sending rate in the last time period, and *avg_latency* is the average delay of the packet in the last time period.

Action. The action space of RL-explore is composed of three discrete actions, namely "×1.25", "×1", and "/1.25".

Among them, "×1.25" means that if this action is selected, the current sending rate will be multiplied by 1.25 and then assigned to the current sending rate. "×1" means that the current sending rate will not be changed, and "/1.25" means that the current sending rate is divided by 1.25 and then assigned to the current sending rate.

Reward. We use the linear reward function to train RL-explore. After the t-th statistical interval is over, the state S_t can be obtained, and the state S_t is input into the neural network. The agent will take an action a_t, and then enter the t+1-th statistical interval. After the t+1th statistical interval ends, the state transitions to s_{t+1}, and the agent can get the reward $r_t = R(S_t, a_t)$ for taking action a_t in the state S_t. The performance of TCP is closely related to congestion control. Different applications have different performance requirements for the network. They are roughly divided into three categories: high throughput, low latency, and low loss. Considering the above three situations comprehensively, we reward throughput, penalty loss and delay when designing RL-explore. The specific reward function is as follows:

$$r = \log throughput_{max} - 10 \times avg\_latency - 10 \times loss \qquad (4)$$

Among them, $throughput_{max}$, *loss* ,*avg_latency* represent the same meaning as in **State.**

ProbeBW. As mentioned earlier, our method is not a pure data-driven RL method, but is similar to the RL method combined with traditional congestion control methods. However, it is not exactly the same as the method mentioned in section 2.3. We are adding some mechanisms in the traditional congestion control algorithm to the RL method. For this article, we have added a bandwidth similar to BBR to the RL method. The mechanism of the detection mechanism. The function of ProbeBW is that after several statistical intervals, if the action selected by the current decision-making network is not to reduce the sending rate, ProbeBW will increase the current send rate by 1.25 times. The advantage of this is that RL-explore can stably detect the maximum bandwidth of the current environment, help it adapt to the network environment with high-speed and dynamic changes in bandwidth, and improve bandwidth utilization.

4.2 Training

We train our agent in simple-emulator, which is an opensource gym. This environment uses a series of parameters to simulate network links. Our model is trained using the PPO algorithm [16].

4.3 Packet-based and RTT-based

Packet-based and RTT-based. In our experiment, the Agent is the sender and its action is to adjust the send rate. To formalize this, we use MIs [17]. MIs refer to the division of time into continuous

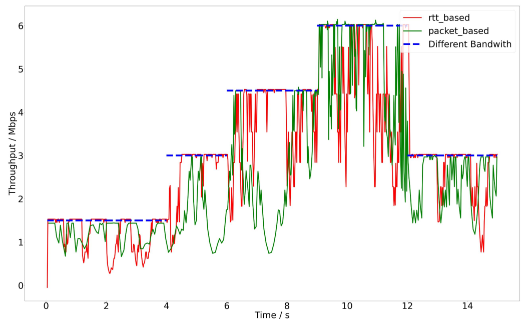

Figure 3: Packet-based and RTT-based Methods in a 15-second Trace of the Throughput on a Link which Alternates between 1.5 and 6 Mbps.

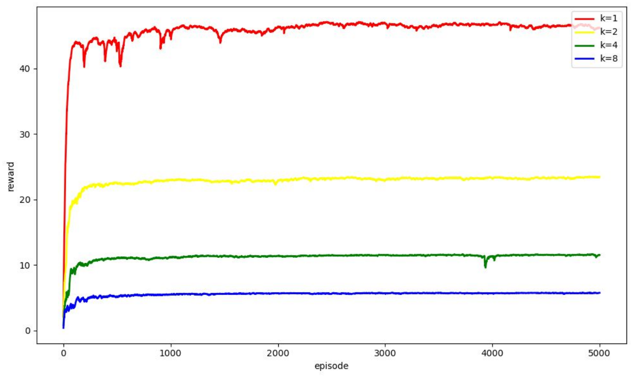

Figure 4: Cumulative Discount Rewards under Different Parameters k.

time periods, called monitoring intervals (MIs), whose length is usually 1 to 2 RTT. In the initial stage of each MIs, the sender can set its sending rate x_t, and then keep it unchanged throughout the MI. The default length of MIs is usually RTT, but our experiment tried two lengths, Packet numbers and RTT. Figure 3 compares the two methods:

Packet-based. The packet number refers to the number of packets ack by the receiver, including all packets including Drop and Finished packets. In our experiments, this packer number is usually set to 50, which means that every time the receiver confirms 50 packets, it represents one MIs.

RTT-based. This method is to use the default length of MIs, we set the length of MIs to 1 RTT. According to our experiments, packet number-based is more sensitive in high-bandwidth environments, it is somewhat slow in low-bandwidth environments, and RTT-based is more stable. This is because in a high-bandwidth network environment, the sender can maintain a higher send rate, and more packets are sent per unit time, which means that the former can make more actions per unit time, while at low in a bandwidth network environment, the number of actions is reduced.

4.4 Model Parameter

Model parameters play a crucial role in training. The details of the parameters in our model will be described in detail below.

Neural network. The neural network of RL-explore consists of an actor network and a critic network. The actor network selects actions according to the input, and the critic network evaluates the current actions. Among them, the actor network is composed of a total of 3 layers of 8×64×64×3, and the critic network is 8×64×64×1.

History length. The history length refers to the data generated after the agent has collected how many actions in the past as the input of the network. K means that the agent makes decisions based on the k latest MIs values of the data. In theory, increasing the history record length should increase performance because it provides additional information. We choose k=1, k=2, k=4, and k=8 for experimental verification in Figure 4. Our experimental results are different from those of Jay N [8], because the delay of the network environment we set is 20ms, which is much larger than that set by Jay N and others. In subsequent experiments, unless otherwise specified, k is selected as 1 by default.

Important super parameter. In RL-explore, the settings of various important hyperparameters are: learning rate $lr = 0.002$, discount factor $\gamma = 0.99$, $\beta = (0.9, 0.999)$.

5 RESULTS

We test the performance of our training model in a variety of network scenarios and compare it with those well-known TCP congestion control protocols. We compare RL-explore with other three typical congestion control algorithms (Reno, Cubic, and pure data-driven RL) in multiple dimensions.

5.1 Bandwidth Sensitivity

Bandwidth sensitivity can show how much an algorithm utilizes bandwidth. An algorithm sensitive to bandwidth changes can adjust its transmission rate more quickly according to the bandwidth in the network, and can provide a higher QoE(Quality of Experience).

Figure 5 shows that under a fixed bandwidth, the throughput of the TCP CUBIC algorithm cannot reach the rated bandwidth in the entire 15s. After TCP RENO reaches the maximum bandwidth, it will drop rapidly. The overall bandwidth utilization rate shows a sharp fluctuation state, only RL- Explorer can reach the rated bandwidth within a few tenths of a second, and the throughput will remain near the rated bandwidth in the following time.

Figure 6 shows the throughput performance of TCP CUBIC and RL-explore under varying bandwidth (the blue dashed line represents the available bandwidth of the link). The throughput of TCP CUBIC is always a certain distance away from the available bandwidth. In the first 4s, even a large throughput fluctuation occurs. In most cases, the throughput of RL-explore is close to the available bandwidth of the link, and it can quickly adapt to changes in network bandwidth. In Figure 6, the throughput of RL-explore far exceeds TCP CUBIC in 80% of the cases, and the throughput is slightly lower than TCP CUBIC in the 12-15 second interval. We can think that RL-explore can maintain the goal of high throughput under varying bandwidths.

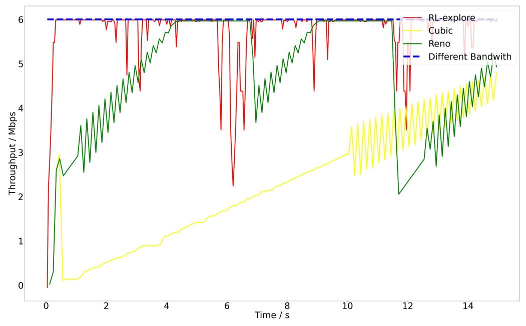

Figure 5: A 15-second Trace of Throughput for TCP CUBIC, TCPRENO and an RL-explore Protocol on a 6 Mbps Bandwidth Link.

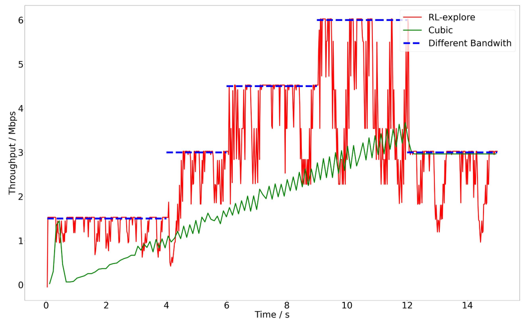

Figure 6: A 15-second Trace of the Throughput of TCP CUBIC and an RL-explore Protocol on a Link which Alternates between 1.5 and 6. Mbps.

5.2 Latency Sensitivity

Figure 7 is a comparison diagram of the latency between TCP CUBIC and RL-explore under the same varying bandwidth. The upper picture is TCP CUBIC, and the lower picture is RL-explore. The blue line in the figure represents the instantaneous latency of the algorithm, the green line represents the average delay generated by the algorithm, and the small red 'x' represents packet loss here.

According to Figure 7, it can be found that the average latency of RL-explore and TCP CUBIC is almost the same, and the instantaneous latency of RL-explore is not as smooth as TCP CUBIC. In addition, there is little difference in packet loss between the two.

5.3 Loss Sensitivity

According to Table 1, in the network environment with varying bandwidth shown in Figure 5, the number of packets sent and confirmed by RL-explore far exceeds the TCP CUBIC algorithm, which means that the throughput of RL-explore is better than that of TCP CUBIC. It is 42% higher, but the packet loss rate of RL-explore is also slightly higher than that of TCP CUBIC.

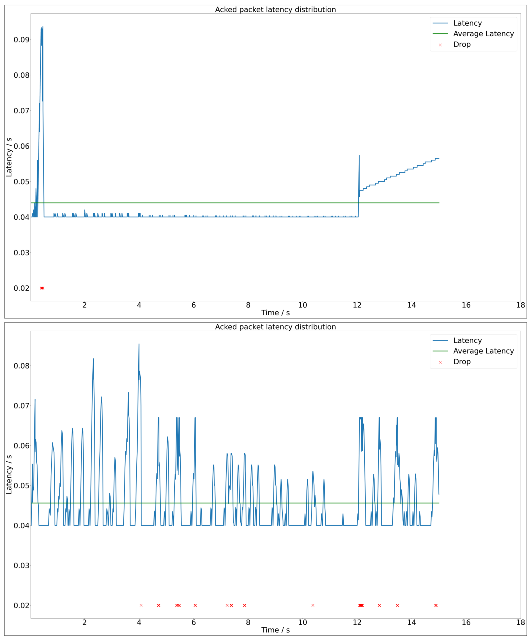

Figure 7: Comparison of TCP CUBIC and RL-Explore Algorithm with Time Delay at Varying Bandwidth as Shown in Figure 5.

Table 1: TCPCUBIC and RL-explore Are the Statistical Averages of Packet Sending in 10 Experiments in the Varying Bandwidth Shown in Figure 5

	Send	Ack	Loss	Loss rate
TCPCUBIC	19918	19854	64	0.32%
RL-explore	28347	28063	284	1%

6 CONCLUSION AND FUTURE WORK

We gave a brief introduction to RL-explore. At the same time, we have conducted multiple comparative evaluations of RL-explore and TCP CUBIC in terms of bandwidth utilization, delay and loss. The results show that RL-explore can also show more performance than TCP CUBIC after limited training.

The reason may be because RL-explore will proactively detect the network bandwidth according to the action and ProbeBW mechanisms, and will learn to keep the sending rate near the rated bandwidth through training. Therefore, it can always maintain a high throughput without increasing the average delay.

There are still some shortcomings in the work of this paper. We only tested it in a simple dumbbell-shaped network topology environment. And the fairness of RL-explore and other TCP algorithms has not been deeply explored, and we will gradually solve these problems in the follow-up work.

REFERENCES

[1] Ha, Sangtae, Injong Rhee, and Lisong Xu (2008). CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS operating systems review, 42(5): 64-74.

[2] Cardwell, Neal, *et al.* (2017). BBR: congestion-based congestion control. Communications of the ACM, 60(2): 58-66.
[3] Jacobson, Van. (1988). Congestion avoidance and control. ACM SIGCOMM computer communication review, 18(4): 314-329.
[4] Sharma, Varun Kumar, Lal Pratap Verma, and Mahesh Kumar (2019). CL-ADSP: Cross-Layer adaptive data scheduling policy in mobile ad-hoc networks. Future Generation Computer Systems, 97: 530-563.
[5] Mnih, Volodymyr, *et al.* (2015). Human-level control through deep reinforcement learning. Nature, 518(7540): 529-533.
[6] Li, Wei, *et al.* (2018). QTCP: Adaptive congestion control with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3): 445-458.
[7] Floyd, S., T. Henderson and A. Gurtov (1999) . The NewReno Modification to TCP's Fast Recovery Algorithm.
[8] Yan, Francis Y., *et al.* (2018). Pantheon: the training ground for Internet congestion-control research. 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18).
[9] Jay, Nathan, *et al.* (2019). A deep reinforcement learning perspective on internet congestion control. International Conference on Machine Learning. PMLR.
[10] Nie, Xiaohui, *et al.* (2019). Dynamic TCP initial windows and congestion control schemes through reinforcement learning. IEEE Journal on Selected Areas in Communications 37(6): 1231-1247.
[11] Abbasloo, Soheil, Chen-Yu Yen, and H. Jonathan Chao (2019). Make tcp great (again?!) in cellular networks: A deep reinforcement learning approach. arXiv preprint arXiv:1912.11735.
[12] Emara, Salma, Baochun Li, and Yanjiao Chen (2020). Eagle: Refining congestion control by learning from the experts. IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE.
[13] Abbasloo, Soheil, Chen-Yu Yen, and H. Jonathan Chao (2020). Classic meets modern: A pragmatic learning-based congestion control for the Internet. Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication.
[14] Sutton, Richard S. and Andrew G. Barto (1999). Reinforcement learning: An introduction. Robotica 17(2): 229-235.
[15] Mnih, Volodymyr, *et al.* (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
[16] Schulman, John, *et al.* (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
[17] Dong, Mo, *et al.* (2015). {PCC}: Re-architecting congestion control for consistent high performance. 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15).