# ZiXia: A Reinforcement Learning Approach via Adjusted Ranking Reward for Internet Congestion Control

Lianchen Jia[1], Tianchi Huang[1], Lifeng Sun[1,2,3*]
[1]*Dept. of CS&Tech.*, [2]*BNRist*, [3]*Key Laboratory of Pervasive Computing, Tsinghua University*
{jlc21@mails., htc19@mails., sunlf@}tsinghua.edu.cn

***Abstract*—Congestion control (CC) algorithms based on deep reinforcement learning (DRL) have shown their great potential to adapt themselves to a variety of network conditions. However, as the real-world network conditions are diverse and the optimization goals for CC are made up of some contradicted metrics, it is hard to balance these contradicted absolute valves, which makes it difficult to faithfully reflect the algorithm performance if only considering transient status as the reward. In this work, we propose a novel DRL-based CC approach ZiXia, which considers the adjusted ranking reward, the long-term relative performance reward adjusted by the transient reward. In detail, we design a virtual algorithm arena including the DRL-agents and other classic algorithms as competitors in the same environment. After these algorithms end, we rank their delay and throughput respectively and combine the two relative rankings as the ranking reward using the special preference. The ranking reward gives us a more flexible and interpretable long-term evaluation method compared with the absolute value of the transient status, which gives us a more intuitive perspective to support multi-objection. To get more fine-grained action rewards, we adjust the ranking reward using a linear combination of transient status. Through various experiments, in simulated environments, ZiXia achieves the highest throughput and reduces 87% delay compared with BBR, and in global real-world environments, ZiXia improves 13% throughput and reduces 3% delay than BBR.**

***Index Terms*—Congestion Control, Deep Reinforcement Learning**

## I. Introduction

Congestion control as a classic core problem in computer networks has been studied for more than 30 years [1]. In these 30 years, many algorithms have been proposed along with the development of networks. However, with the increase of the new network infrastructures such as 5G and transcontinental networks, the network conditions are becoming more and more complex [2], and new application scenarios such as live broadcasts and mobile cloud conferences put forward new demands about throughput and delay. Traditional heuristic algorithms use packet loss or delay as the signal of congestion. However, under complex network conditions, it is difficult to find an appropriate heuristic function to handle all situations, such as wireless networks with random loss and transcontinental networks with large packets queue buffer.

Learning-based algorithms are proposed (such as Remy [3], Indigo [4], PCC [5] [6] [7]) in this context, which do not require too much manual engineering and have the potential to fit various conditions. Deep reinforcement learning performs well on many complex reality tasks [8], and recently many CC algorithms based on DRL methods (such as Aurora [9], Eagle [10], Orca [1]) have achieved excellent performance under various network conditions.

These DRL algorithms obtain transient rewards by interacting with the environment and move towards the higher cumulative sum of transient rewards. However, it is hard to get the proper transient reward function to truly reflect the performance as there is a conflict between the throughput and delay. An algorithm that prefers high throughput might cause a large delay as it occupies the packets queue buffer and another algorithm that prefers low latency might obtain low throughput as its conservative strategy does not make full use of bandwidth. So, the linear combination must balance these two conflicting variables but this is not an easy job. We can hardly give a proper reward function that can perfectly fit any network conditions as the diversity of the real-world network conditions, and an inaccurate reward function will affect the stability and the performance of the algorithm [11].

To deal with the difficulty of designing rewards, we use the ranking reward to evaluate the algorithm performance. We use the relative value compared with other algorithms instead of the absolute value to measure the performance. This method avoids the deviation of the reward value caused by the different network environments. No matter in any network scenario, the algorithm's throughput ranks high indicating that it performs well on this aspect. We can also design special ranking reward rules to meet special needs.

Although the ranking seems to be a great way to evaluate performance, there is a shortcoming that cannot be ignored. The ranking reward is too sparse which makes it difficult for the agent to learn. So, the transient reward is still necessary.

Therefore, we propose ZiXia, a novel DRL-based CC approach, which considers adjusted ranking rewards. We use the ranking with other algorithms under the same network conditions to measure their long-term performance. We rank throughput and delay separately and then combine them linearly using specific rules to meet diverse application requirements. We also use transient status to adjust the ranking reward which provides more fine-grained action rewards.

We compare the performance of ZiXia with other state-of-the-art CC algorithms in both simulated environments and real-world environments. In the simulated environments, ZiXia performs much better than most algorithms in various network conditions, and in the global real-world environments we deploy 8 server nodes around the world and the performance of ZiXia is stable and competitive.

In general,we summarize the contributions as follows:

- We propose ZiXia, a DRL-based CC approach that uses adjusted ranking reward. We design a virtual algorithm arena to get the ranking reward and consider transient reward to adjust it.
- We test ZiXia in both the simulated network environments and the real network environments, and we find it achieve outstanding performance in both cases compared with recently proposed algorithms.

## II. Motivation

### A. Why need the ranking reward in DRL-based CC?

The answer is the reward of the DRL is too difficult to design. Although the DRL-agent automatically moves towards higher rewards to reduce manually engineering, the reward itself is hard to define. As shown in the introduction, these two conflicting parameters (throughput and delay) are difficult to balance in various network conditions.

Just give a simple example, one algorithm A gets 50Mbps throughput and 100ms delay when the network capacity is 100Mbps, and another algorithm B gets 1.2Mbps and 100ms when the network capacity is 1.3Mbps. If we use the linear combination of throughput and delay as the reward, such as Eq. 1,

$$r = throughput - delay \tag{1}$$

the reward of algorithm A is much larger than the latter and the agent will consider the former is much better. In the gradient ascent for deep reinforcement learning, the agent's strategy will move much closer to the former and the "smart" agent will learn to send 50Mbps, which is a disaster for the latter's network environment. It seems it is impossible to judge an algorithm's performance independently.

The ranking reward gives us another perspective about the algorithm performance, which uses relative value instead of absolute value. It evaluates the algorithm not by how much throughput and delay it has achieved, but by comparison with other algorithms under the same network conditions. This evaluation method avoids the bias of the DRL strategy caused by the excessive reward value brought by the different network environments. In addition, it is easier to adjust preference for throughput and delay using ranking, which gives us a simple way to meet diverse needs. For example, you can set a reward to represent that it is enough for throughput beyond half of the competitors, but the delay ranking should be as small as possible, which is hard to design using the linear combination of transient status but is easy for ranking-based.

### B. How to use the ranking reward in DRL-based CC?

It seems that the ranking is a great method to evaluate the algorithm performance, but it is not a good idea to use it directly as a reward. Long-term performance ranking is easy to get but it is hard to get transient ranking. At time $t$, the throughput obtained by the agent is not the result of the current sending rate, which makes it difficult to compare transient action. So the ranking reward is the sparse long-term reward.

Although the accurate transient reward is hard to design, the core idea of the transient reward is to guide the algorithm to achieve high throughput and low delay, which is similar to our ranking reward. So we can consider transient reward as a kind of auxiliary task reward [12] [13], which gives us the fine-grained perspective to reflect the actions of the agent. So we still consider the transient reward to adjust the ranking reward.

## III. Design of ZiXia

In this section, we describe the proposed system ZiXia in detail. ZiXia's basic structure is illustrated in Figure 1. The arena block provides the ranking reward, and the transient block interacts with the environment and provides transient data. Then the learning block adjusts the ranking reward using the transient data and feed these new data to DRL-agents.

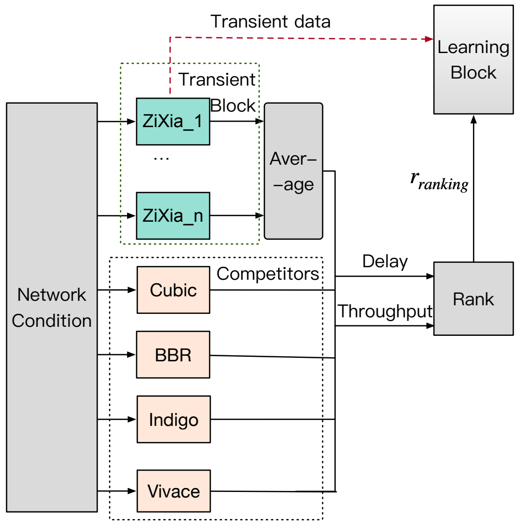


Fig. 1. ZiXia's System Overview

### A. Arena Block

It is hard to evaluate the algorithm independently, so we construct an algorithm arena to use the ranking compared with other algorithms in the same network condition to evaluate performance. In detail, we use multi-threads to construct this as Figure 1. To improve the sampling efficiency of interaction with the environment, we use multiple threads to run the DRL algorithm. Therefore, there is not much difference in the training efficiency compared with the pure distributed reinforcement learning algorithm [14]. To get competitive and diverse competitors, we carefully selected 4 classic algorithms recently proposed, which include BBR [15], Cubic [16], Vivace [6], Indigo. BBR and Cubic are heuristic-based algorithms which widely deployed on the internet. Vivace is an online-learning-based algorithm that uses the transient utility to adjust its sending rate. Indigo is an imitation-learning-based algorithm and behaves very well in some conditions.

TABLE I
STATE OF THE TRANSIENT BLOCK

| | |
|---|---|
| $delay_{ewma}$ | the ewma number of queue delay |
| $deli\_rate_{ewma}$ | the ewma number of the receiver received rate |
| $send\_rate_{ewma}$ | the ewma number of the sender send rate |
| $cwnd$ | the current congestion window |
| $duration$ | the time from start until now |

Every round of training, each thread runs 30s in the same network condition, whose parameters include delay, loss ratio, length of the queue buffer, and bandwidth traces. After all of them end, we calculate the throughput and the 95th percentile per-packet one-way delay. Using multi-threads, we obtain some different values of DRL-agents and calculate their average value to eliminate accidental errors. To meet the real needs of different applications, we rank these two sets of values separately in ascending order and use different weights ($\alpha$ and $\beta$) for the linear combinations of delay ranking $rank_{del}$ and throughput ranking $rank_{tput}$ as ranking reward $r_{ranking}$. (See in Eq. 2).

$$r_{ranking} = \alpha * rank_{tput} - \beta * rank_{del} \tag{2}$$

## B. Transient Block

**Input** Although the network is partially observable, we hope to provide the agent with as much useful information as possible to help the agent understand the environment. As mentioned in [9], increasing the history length of the state would increase the performance of the agent, so in this work, we choose the past five states as history information to balance the trade-off between the performance and the training cost. The agent updates the state when it receives an ACK, and makes a decision every 10ms. At time $t$, the state $s_t$ can be represented as Table I.

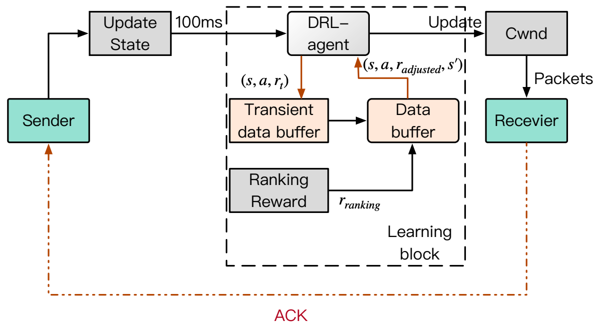


Fig. 2. ZiXia's Training Workflow

In detail, $deli\_rate_{ewma}$ indicates the EWMA (Exponentially Weighted Moving-Average) of the delivery rate which reduces the impact of too larger or too small values, and the $send\_rate_{ewma}$ is the EWMA of the send rate. To get the more accurate delay, we calculate the delay as $delay = RTT - min\_rtt$ and update the $min\_rtt$ when these algorithms run. We normalize them respectively to deal with the difference in the order of magnitude.

**Output** The state-space of the cwnd itself is too large as the network conditions vary greatly in reality. So, it would be a disaster for reinforcement learning if we directly use cwnd as the action space. In this work, we picked 7 values in the [-1,1] as the action for the algorithm. At time $t$, the $cwnd_{t+1}$ will be updated from $cwnd_t$ by the current action $a_t$ following the function:

$$cwnd_{t+1} = cwnd_t * (1 + a_t) \tag{3}$$

**Transient Reward** Previous work spends a lot of time optimizing reward functions [1] [10], but in our work, the transient reward is still necessary but does not need much engineering work. We just use the simplest linear combination of delay and throughput, which can be represented as

$$r_t = 0.1 * throughput - 100 * delay \tag{4}$$

The unit of throughput is $Mbps$ and delay's unit is $s$. In this setting, the $r_t$ plays a supporting role compared with $r_{ranking}$.

## C. learning block

**Adjusted Reward** The typical interaction process of reinforcement learning is that at time $t$, the agent first observes the state of the environment $s \epsilon S$, and then selects action $a$ based on the strategy $\pi(s)$ and action $a$ acts on the environment to cause the environment state to change to $s'$, and at the same time, the agent receives the reward $r$ [8].In our work,we obtain the $r_t$ immediately in transient block and store the $(s, a, r_t, s')$ in the transient data buffer.For arena block, the agent receives the $r_{ranking}$ after it ends, and all states and the actions on this trace share the same reward.

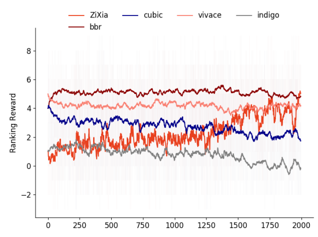
In our framework, the tuple of the $(s, a, s')$ is the same in both transient block and arena block as these data are produced by the same trace and the same interaction. So, we can use transient reward to adjust the ranking reward and define the $r_{adjusted}$ as:

$$r_{adjusted} = r_{ranking} + \gamma r_t \tag{5}$$

We use $\gamma$ to balance these rewards. In this way, we get the data $(s, a, r_{adjusted}, s')$ which used for agent to train.

**Dual-PPO and Adaptive Entropy Weight Decay** As the state space of the congestion control is large which makes it difficult for reinforcement learning to converge to a stable value, the Dual-PPO [17] is used in our work. The probability ratio $p_t(\theta)$ denotes the probability ratio between current policy $\pi_\theta(a_t|s_t)$ and the old policy $\pi_{old}(a_t|s_t)$. It can be extremely large, which may lead to an excessively large policy deviation. So the standard PPO [18] involves a ratio clip which reduce the influence of extreme values. However,when the $\pi_\theta(a_t|s_t) \gg \pi_{old}(a_t|s_t)$ and advantage value $\hat{A}_t < 0$, we can see $p_t(\theta)\hat{A}_t \gg 0$ and this may also cause excessively large policy deviation. So in the Dual-PPO, if the $\hat{A}_t > 0$, it will work equal to the standard PPO algorithm and if the $\hat{A}_t \leqslant 0$, Dual-PPO will clip the $p_t(\theta)$ with the lower bound of the $\hat{A}_t$.

The value of entropy affects the speed and performance of reinforcement learning training. To alleviate this issue, we use the adaptive entropy weight decay (AEWD) [11] to dynamically adjust the entropy weight.


(a) ZiXia's Ranking Reward

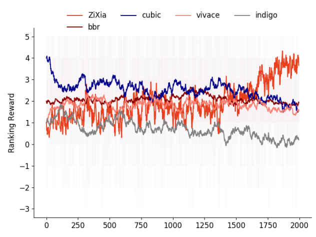

(b) ZiXia_delay's Ranking Reward

Fig. 3. The Change of Ranking Reward

TABLE II
PARAMETERS. DURING THE TRAINING.

| average trace capacity | delay | loss | queue |
|---|---|---|---|
| 10.33Mbps | 1-100 ms | 0%-15% | 20-1000 packets |

## IV. EVALUATION

We use mahimahi [19] to simulate the real network environments which include the loss ratio, delay, queue length, and the capacity of the link. Table I list the network parameters used in the training. To improve the efficiency of the training, we use 4 threads for the classic algorithm to compare and 12 for DRL-agents that are used to collect data. We train ZiXia on a 16-core server, and each thread runs for 30s under the same network conditions. Then we calculate the throughput and 95th delay to rank them respectively. We set the $\alpha$=2,$\beta$=1 in Eq. 2 to calculate the $r_{ranking}$, which means we consider it is cost-effective to improve throughput rankings at the cost of a bit delayed ranking. After getting the $r_{ranking}$, we modify the data buffer collected by agents to get the adjusted data using Eq. 5. Many previous work focus on optimal the $\gamma$ in Eq. 5 [20] [21], but in our work, we just set the $\gamma$ as 1. According to Eq. 4, the $r_t$ must be less than the $\frac{1}{10}$ of the throughput, and the average throughput in training is only 10.33Mbps, so the ranking reward plays a leading role.

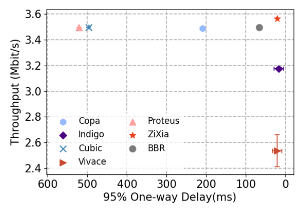
As Figure 3(a), ZiXia achieves the highest ranking reward after about 2000 iterations. We test its performance in both simulated environments and real-world environments. To show the ability to support special preference, we design another algorithm ZiXia_delay, whose ranking reward in training as Figure 3(b) and more details will be discussed below.

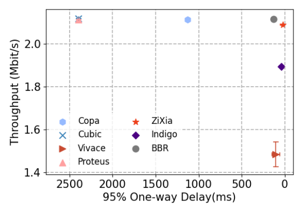
### A. Testbed and Baselines

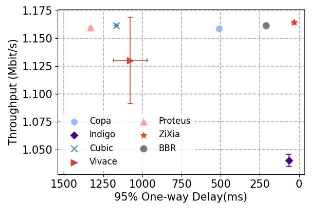
We take 8 state-of-the-art CC algorithms published in recent years as the baselines. Three of them are the heuristic-based algorithms, BBR (model-based), Cubic (loss-based) and Copa (delay-based) [22]. For the learning-based algorithms, we select Indigo (imitation-learning-based), Eagle (DRL-based), Orca (hybrid of heuristic-based and DRL-based). And we also choose the online-learning-based algorithms, Vivace and Proteus [7].

TABLE III
RESULTS OF REAL NETWORK TRACE

| algorithm | throughput/Mbps | delay/ms |
|---|---|---|
| ZiXia | 1.43 | 50.91 |
| Cubic | 1.42 | 1917.40 |
| Proteus | 1.42 | 2025.77 |
| Copa | 1.42 | 756.02 |
| BBR | 1.42 | 383.73 |
| Indigo | 1.29 | 122.17 |
| Vivace | 1.20 | 1628.24 |


(a) test_norway_bus


(b) test_norway_car


(c) test_norway_tram

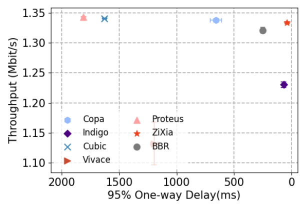

(d) test_norway_ferry

Fig. 4. The performance of Real Network Trace

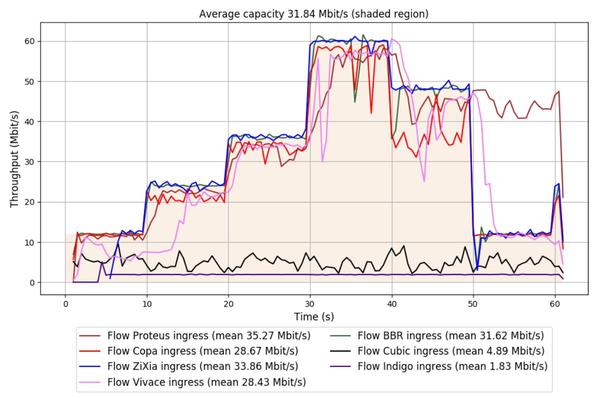
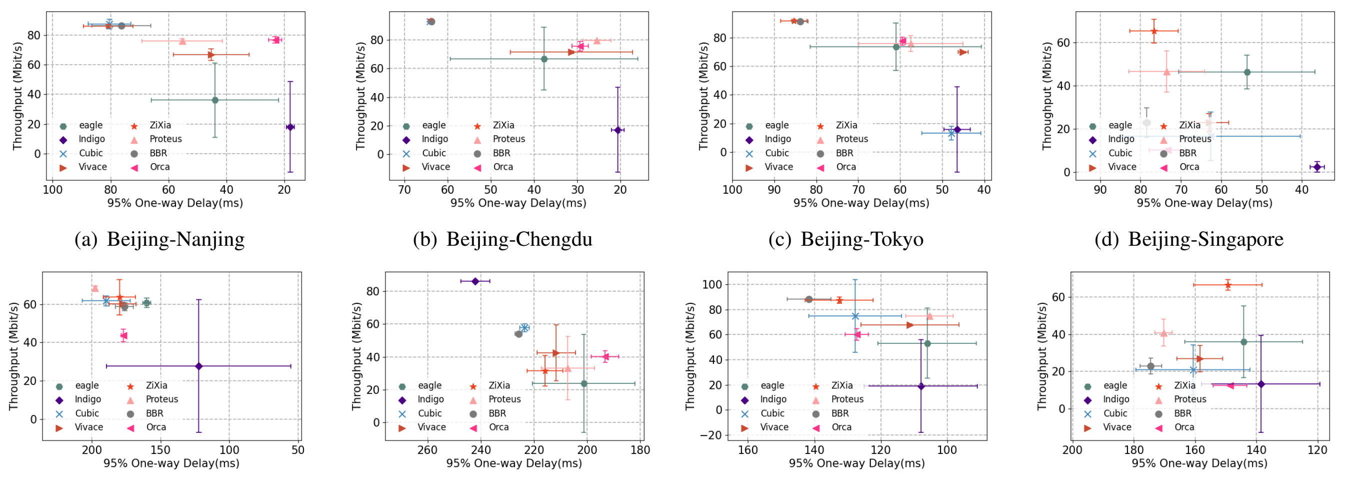


Fig. 5. The Performance in Synthetic Trace

### B. Simulated environments

**Real Network Trace** We use the reality trace from FCC [23] and fix the queue buffer at 100 packets to test the performance of ZiXia. We use eight of them which does not appear in the training set and to reduce accidental errors, we run every algorithm 5 times and calculate the average standard deviation which uses an error bar to represent. We show some results in Figure 4, and Table III shows average throughput and delay in this experiment. In Figure-4(b) and Figure-4(d), we observe that the online-based CC algorithms are not suitable for these dynamic bandwidths, which does not give these algorithms enough time to converge. Cubic achieves high throughput but meets large delay as Cubic uses packets loss as congestion signal and chooses to occupy the queue buffer first. ZiXia achieves the highest throughput and reduces 87% delay compared with BBR.

**Synthetic Trace** To better display the details of the algorithm execution process, we synthesize one trace whose capacity is from 12Mbps to 60Mbps. This trace changes the network bandwidth every 10s, and to meet the reality we add 10ms delay and 1% random loss using mahimahi. Figure-5 shows the details of each algorithm. We observe that Cubic and Indigo fail in this experiment. The reason for Cubic is it



(a) Beijing-Nanjing (b) Beijing-Chengdu (c) Beijing-Tokyo (d) Beijing-Singapore

(e) Beijing-Frankfurt (f) Beijing-Moscow (g) Beijing- Silicon Valley (h) Beijing-Virginia

Fig. 6. The Performance of Real-World Environments

can not figure out random loss or congestion loss. For Indigo, the imitation-learning-based CC, there is a big gap between this experiment and its training environment. Proteus achieves the highest throughput, but the large delay is unacceptable. Vivace meets similar conditions, as these online-based CC algorithms need more time to converge. Copa, BBR, and ZiXia perform well in this experiment, but the first two algorithms lack the ability to adapt quickly when the network changes rapidly. At the 40s point of the trace, the capacity changes from 60Mbps to 48Mbps, Copa and BBR fail to adjust their strategy. At this point, the queue buffer is full of packets as the network capacity decreases and Copa suffers the performance degradation for the increase of delay. For BBR, the detection of the environment is delayed, which can not give BBR the current network environment status in time. ZiXia performs best in this experiment, which quickly adapts to network bandwidth changes. This experiment shows the detail of these algorithms and ZiXia performs best and demonstrates the ability to quickly adapt to changes in the network.

## C. *Real-World environments*

To verify the performance of our algorithm in real scenarios, we deployed our algorithm on eight nodes around the world and compare its performance with recent work. For every test, we send a flow from Beijing and last 30s, and each algorithm runs 5 times to reduce the accidental error. We show results in Figure 6, and calculate their average throughput and delay in Table V. Inspired by previous work [1], we divide these servers into two groups, 4 inter-continental nodes and 4 intra-continental nodes. In these experiments, we add recent state-of-the-art DRL-based CC algorithms (Orca and Eagle) to test.

**Intra-Continental Scenarios** In intra-continental scenarios, we have two servers in china and two servers in Tokyo and Singapore. Intra continental network usually has low link delay and we observer that the delay of all algorithms is less than 100ms. We get the highest throughput in these four servers and we are significantly better than other algorithms in the Singapore node. Eagle performs unstable with large variance at each node especially in terms of delay. Orca performs well in most servers but fails in the Singapore node which only gets half of ZiXia's throughput.

TABLE IV
LOCATION OF SERVERS USED IN REAL-WORLD ENVIRONMENTS.

| Continent | City, Country | |
|---|---|---|
| Asia-1 | Nanjing, China | Chengdu,China |
| Asia-2 | Tokyo, Japan | Singapore |
| Europe | Frankfurt, German | Moscow, Russia |
| North America | Silicon Valley, United States | Virginia, United States |

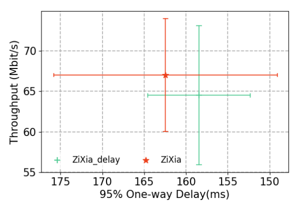
TABLE V
RESULTS OF REAL-WORLD ENVIRONMENTS

| algorithm | throughput/Mbps | delay/ms |
|---|---|---|
| ZiXia | 73.22 | 123.02 |
| BBR | 64.66 | 127.57 |
| Proteus | 61.91 | 111.55 |
| Vivace | 53.55 | 105.45 |
| Cubic | 53.19 | 119.57 |
| Orca | 49.57 | 103.95 |
| Eagle | 49.55 | 100.00 |
| Indigo | 28.37 | 91.60 |

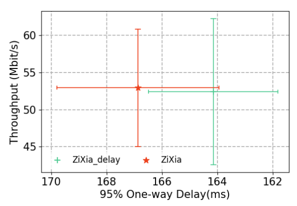
**Inter-Continental Scenarios** Inter-continental network environments are more complicated as they need to transmit longer distances which use different strategies and network structures [1]. This makes it difficult for algorithms optimized for specific links to work well under such network conditions. In these scenarios, ZiXia is still competitive compared with the recent algorithm which achieves the highest throughput on three of the four servers. Eagle still behaves unstable, especially in Moscow. Orca performs not well except for Moscow and we can observe that it does not achieve throughput higher than 60Mbps. Indigo performs great in Moscow but fails in other servers as there is a big gap between the real network environment and the environment in imitation learning.

**Multi-objection** The ranking reward gives us a more interpretable algorithm evaluation method compared with the previous work [24], which makes it simple to support multi-objection. We get another algorithm ZiXia_delay, using different ranking reward as :

$$r_{ranking} = \begin{cases} 2 * rank_{tput} - rank_{delay}, if\ rank_{tput} < 3 \\ 6 - rank_{delay}, else \end{cases} \tag{6}$$



(a) Beijing-Virginia



(b) Beijing-Toronto

Fig. 7. The Performance of Multi-Objection

This means we think it is enough for throughput beyond other there algorithms in this arena, but the delay ranking should be as small as possible. In Figure 3(b), ZiXia_delay after about 2000 iterations significantly exceeds other algorithms in ranking reward. Compared with Figure 3(a), we find classic algorithms perform well under simple preference, but under complex preference, such as Eq. 6, our algorithm is significantly better than them which shows our ability to accurately reflect specific preferences. We also test the ZiXia_delay and ZiXia in real-world scenarios as Figure 7. We respectively run it 5 times from Beijing to Virginia and Toronto, and calculate their mean and variance. We can see that in each scenario, the performance of ZiXia_delay is close to the ZiXia, but the former gets lower delay as our preference.

## V. Conclusion

We presented ZiXia, a DRL-based CC algorithm using the adjusted ranking reward. ZiXia introduces this long-term interpretable performance evaluation method into DRL-based CC algorithms which avoids the difficulty in reward design and also considers transient perspective to alleviate the problem of sparse rewards. This relative evaluation method gives us a more flexible way to support the special complex preference for throughput and delay. We have done a lot of experiments in the simulated environment and global real-world environment, and ZiXia has achieved great performance compared with the recent state-of-the-art algorithm in all experiments.

## Acknowledgment

We thank the anonymous ICC reviewers for their constructive feedback. This work was supported by NSFC under Grant 61936011, 61521002, Beijing Key Lab of Networked Multimedia.

## References

[1] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 632–647.

[2] W. Wei, H. Gu, and B. Li, "Congestion control: A renaissance with machine learning," *IEEE Network*, 2021.

[3] 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.

[4] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *2018 {USENIX} Annual Technical Conference* (*$USENIX ATC$ 18)*, 2018, pp. 731–743.

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

[6] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "{PCC} vivace: Online-learning congestion control," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 343–356.

[7] T. Meng, N. R. Schiff, P. B. Godfrey, and M. Schapira, "Pcc proteus: scavenger transport and beyond," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 615–631.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

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

[10] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 676–685.

[11] T. Huang, R. Zhang, and L. Sun, "Zwei: A self-play reinforcement learning framework for video transmission services," *IEEE Transactions on Multimedia*, 2021.

[12] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[13] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, pp. 3675–3683, 2016.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[15] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[16] 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.

[17] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Q. Chen, Y. Yin, H. Zhang, T. Shi, L. Wang, Q. Fu, W. Yang, and L. Huang, "Mastering complex control in moba games with deep reinforcement learning," 2020.

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

[19] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for {HTTP}," in *2015 {USENIX} Annual Technical Conference (ATC 15)*, 2015, pp. 417–429.

[20] Y. Du, W. M. Czarnecki, S. M. Jayakumar, M. Farajtabar, R. Pascanu, and B. Lakshminarayanan, "Adapting auxiliary losses using gradient similarity," *arXiv preprint arXiv:1812.02224*, 2018.

[21] X. Lin, H. S. Baweja, G. Kantor, and D. Held, "Adaptive auxiliary task weighting for reinforcement learning," *Advances in neural information processing systems*, vol. 32, 2019.

[22] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 329–342.

[23] M. F. B. Report, "Raw data measuring broadband america 2016," https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016, [Online; accessed 19-July-2016].

[24] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," *arXiv preprint arXiv:2107.01427*, 2021.