RAN Information-Assisted TCP Congestion Control Using Deep Reinforcement Learning With Reward Redistribution

Minghao Chen, *Graduate Student Member, IEEE*, Rongpeng Li, *Member, IEEE*, Jon Crowcroft, *Fellow, IEEE*, Jianjun Wu, Zhifeng Zhao, *Member, IEEE*, and Honggang Zhang, *Senior Member, IEEE*

***Abstract*— In this paper, we aim to propose a novel transmission control protocol (TCP) congestion control method from a cross-layer-based perspective and present a deep reinforcement learning (DRL)-driven method called DRL-3R (DRL for congestion control with Radio access network information and Reward Redistribution) so as to learn the TCP congestion control policy in a superior manner. In particular, we incorporate the RAN information to timely grasp the dynamics of RAN, and empower DRL to learn from the delayed RAN information feedback potentially induced by several consecutive actions. Meanwhile, we relax the implicit assumption (that the feedback to one specific action returns at a round-trip-time (RTT) after the action is applied) in previous researches, by redistributing the rewards and evaluating the merits of actions more accurately. Experiment results show that besides maintaining a reasonable fairness, DRL-3R significantly outperforms classical congestion control methods (e.g., TCP Reno, Westwood, Cubic, BBR and DRL-CC) on network utility by achieving a higher throughput while reducing delay in various network environments.**

***Index Terms*— Deep reinforcement learning, congestion control, radio access network, reward redistribution, delayed feedback.**

Manuscript received December 1, 2020; revised May 5, 2021 and August 30, 2021; accepted October 16, 2021. Date of publication October 26, 2021; date of current version December 17, 2021. This work was supported in part by the National Key R&D Program of China under Grant 2020YFB1804800, in part by the National Natural Science Foundation of China under Grants 61731002 and 62071425, in part by the Zhejiang Key Research and Development Plan under Grants 2019C01002 and 2019C03131, in part by the Huawei Cooperation Project, in part by a Project sponsored by the Zhejiang Lab under Grant 2019LC0AB01, in part by a Project sponsored by the Ministry of Industry and Information Technology under Grant 2019-00891-2-1, and in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LY20F010016. A part of this paper has been accepted by the 2021 IEEE International Conference on Communications Workshop [1] [DOI: 10.1109/ICCWorkshops50388.2021.9473523]. The associate editor coordinating the review of this article and approving it for publication was M. C. Gursoy. *(Corresponding author: Rongpeng Li.)*

Minghao Chen, Rongpeng Li, and Honggang Zhang are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: minghaochen@zju.edu.cn; lirongpeng@zju.edu.cn; honggangzhang@zju.edu.cn).

Jon Crowcroft is with the Department of Computer Science, University of Cambridge, Cambridge CB2 1TN, U.K. (e-mail: jon.crowcroft@cl.cam.ac.uk).

Jianjun Wu is with Huawei Technologies Company, Ltd., Shanghai 201206, China (e-mail: wujianjun@huawei.com).

Zhifeng Zhao is with the Zhejiang Lab, Hangzhou 311121, China (e-mail: zhaozf@zhejianglab.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2021.3123130.

Digital Object Identifier 10.1109/TCOMM.2021.3123130

I. Introduction

THE unprecedented growth of Internet-based applications has put significant strains on the Internet. Due to excessively high data-sending rates or poor-quality channels, congestion could happen at the bottleneck of Internet. As one of the most important component of transport layer (TL), congestion control has become a feasible solution to this problem by trying to achieve a subtle state of equilibrium between congestion avoidance and utilization improvement, and many congestion control methods have been proposed in recent decades. However, existing methods still have some serious shortcomings. Firstly, some fundamental assumptions therein are impractical in real-world networks, such as the assumption that there is only one segment loss in each fast retransmission process (e.g., TCP Reno [2]). Secondly, most methods are rule-based so that they can only take actions following some pre-set rules, which usually causes the failure to adapt to underlying changes in networks.

To solve the aforementioned issues, deep reinforcement learning (DRL) has been proposed in TCP congestion control, since DRL can learn how to interact with the environment (i.e., the Internet) without prior knowledge and gradually find policies to obtain higher reward [3], [4]. The advantages of applying DRL in congestion control are two-folded. On the one hand, DRL can better fit the dynamic feature of congestion control, because it can learn the dynamic changes of Internet based on its experience so as to find superior congestion control policies. On the other hand, DRL doesn't need to make excessive assumption on the dynamic feature or architecture of Internet. For example, it doesn't limit the number, location and access mode (wired or wireless) of bottleneck. Unfortunately, previous researches on DRL-based congestion control still possess some shortcomings. Firstly, many previous researches treat radio access network (RAN) the same as wired network, thus making the agent unable to fully utilize the feature and information of wireless network. In this regard, the idea to include RAN information in congestion control could be re-leveraged [5], [6]. Secondly, as the feedback to each controlling action is delayed, some previous researches have implicitly set an assumption, i.e., the feedback of each action

returns at a specific interval (e.g. an RTT) after the action is applied, which may not hold in many practical TCP congestion control scenarios. In addition, the acquired feedback may be induced by several consecutive actions and reflect their joint effect. Hence, it becomes a necessity to infer the "true" reward (i.e., *redistributed reward*) for each action redistributed from the acquired delayed feedback (i.e., *delayed reward*[1]). Thirdly, it is rather difficult for a single agent to control the data-sending rate of several TCP flows simultaneously.

In this paper, we focus on the congestion control problem in an RAN-information assisted network and develop a DRL-based single-agent congestion control algorithm named **DRL-3R** (**DRL** for congestion control with **R**AN information and **R**eward **R**edistribution). Generally speaking, DRL-3R firstly collects the RAN information piggybacked with ACK segment, and predicts the current RAN information from the delayed one(s). Afterwards, DRL-3R controls the data-sending rate (i.e., the *action*) with both the predicted RAN information (i.e., the *RAN state*) and other information (i.e., the *Transport Layer state*) collected by the server. Meanwhile, DRL-3R stores the historical state, action and delayed reward, and analyzes the delayed reward to get the redistributed reward for each action at the end of each episode. Finally, it learns from the historical experience to yield superior performance, achieving higher throughput while reducing RTT, and maintaining excellent fairness. Considering the latest 5G and its feature, i.e., high speed and low latency, DRL-3R can better utilize the advancement of 5G, so as to provide better quality of experience. Moreover, DRL-3R assumes that RAN is not always a bottleneck of transmission, which is consistent with the development of 5G, since advancement in wireless access technologies has made wireless transmission more stable and faster. Contributions of this paper[2] are summarized as follows.

- We incorporate RAN information in DRL-based congestion control, which facilitates the agent to know the information of a potential bottleneck directly and take actions to prevent congestion in RAN, and increases the network utilization eventually. We also propose a method to predict latest RAN information from the delayed RAN information.
- We propose a method to relax the implicit assumption in previous DRL-based congestion control methods. Specifically, we relax the assumption in two aspects: (1) we assume the feedback returns *around* (instead of *at* or *before*) a specific time interval; (2) we acknowledge that each delayed reward contains the feedback of several consecutive actions, and try to redistribute the delayed reward to get the feedback of each action. To realize this goal, we specially design the utility and reward function, and redistribute delayed rewards to each action inspired by RUDDER [7]. We believe this contribution could be widely applied in a number of communications and networking scenarios with commonly delayed feedback for consecutive actions.
- To effectively control the data-sending rate of all TCP flows with one agent, we design a representation network to extract global information at each time-step. Meanwhile, we calibrate the action-selection process so as to obtain superior performance in terms of fairness.
- We compare the performance of DRL-3R with several widely-known baselines (e.g., TCP Reno [2], Westwood [8], Cubic [9], BBR [10] and DRL-CC [11]) and demonstrate the effectiveness and robustness of DRL-3R in terms of fairness, throughput and RTT.

The remainder of this paper is organized as follows. We firstly discuss related works in Section II. In Section III, we present the system model and formulate the problem. In Section IV, we present the architecture of DRL-3R in details. We evaluate the performance of DRL-3R in Section V and conclude this paper in Section VI.

II. Related Works

A. TCP Congestion Control

Congestion is a "jam" situation of the network, which could happen if all network participants send data too fast or the channel quality is poor. Traditionally, congestion control methods, which aims to prevent congestion, takes segment loss or bandwidth estimation as reference to control the congestion windows (i.e., *cwnd*). In recent years, researchers try to include RAN information in congestion control. In 2015, Feng *et al.* proposed CQIC [5], which estimates the bandwidth from the average channel quality indicator (CQI) of RAN and then adjusts sending rate. In 2020, Xie *et al.* proposed PBE-CC [6], which assumes that clients can get RAN information in real time and estimate bottleneck bandwidth accurately. PBE-CC shows excellent performance on throughput while reducing RTT in several scenarios. However, both methods need clients or users to estimate the available bandwidth or data rate accurately based on pre-defined rules, which may be hard to be implemented in real environment.

B. Deep Reinforcement Learning

Reinforcement learning (RL) is adopted to solve problems formulated as Markov decision process (MDP). Typically, there is an agent working in a specific environment. At each time-step t, the agent receives current state $\mathbf{s}_t$ in state space $\mathcal{S}$, takes action $\mathbf{a}_t$ in action space $\mathcal{A}$ according to a policy $\pi(\mathbf{s}_t)$, and receives the reward R_{t+1} at the next time-step. The target of RL is to learn how to map state to actions at each time-step to maximize the reward [12].

A crucial component of RL is value function, which is an estimation on how much accumulated discounted reward the agent can receive in following time-steps. We mainly focus on the state-action value function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ (also called

[1]Note that in classical reinforcement learning perspective, based on the Markov property, only the latest state-action pair contributes to the reward returned by the environment. Instead, the "delayed" reward is induced by not only the latest state-action pair, but also previous ones. Hence, the terminology "delayed" is slightly abused.

[2]Notably part of the paper has been accepted in [1]. Compared with it, this paper provides more detailed description on DRL-3R and adds significant content to analyze reward redistribution more rigorously, thus proving its correctness and rationality in reinforcement learning perspective. Also, this paper evaluates DRL-3R in more complicated environments, proving more solid basis and evidence on its effectiveness. Furthermore, it discusses the effectiveness on RAN information, reward redistribution and parameters of DRL-3R and provides additional implementation details.

Q-value) [12], which can be defined as $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1}]$, where π is the adopted policy and γ is the discount factor limited in $(0, 1]$.

In recent years, the development of deep learning has inspired researchers to consider using deep neural network (DNN) as function approximator. In 2016, Mnih *et al.* proposed a value-based Q-learning-based DRL method, i.e., deep Q-learning (DQL) [13], in which the DNN works as an approximator to calculate value function $Q(\mathbf{s}_t, \mathbf{a}_t)$.[3] During training, for each experience $(\mathbf{s}_t, \mathbf{a}_t, r_{t+1}, \mathbf{s}_{t+1})$, the target value y_t can be calculated from Bellman Equation [12] as

$$y_t = r_{t+1} + \gamma \cdot Q(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1})|\boldsymbol{\theta}_Q) \tag{1}$$

where π is the policy defined as $\pi(\mathbf{s}_t) = \arg\max_{\mathbf{a}}(Q(\mathbf{s}_t, \mathbf{a}|\boldsymbol{\theta}_Q))$ and $\boldsymbol{\theta}_Q$ is the parameter. The DNN in DQL (also called DQN) can be trained using mean square error (MSE) between $Q(\mathbf{s}_t, \mathbf{a}_t)$ and y_t. Similar to Q-learning, DQL can only choose an action from $\mathcal{A}$ with finite elements (i.e., discretized actions). To bring continuous actions in RL, researchers have proposed policy-based and actor-critic DRL methods. In 2016, Lillicrap *et al.* proposed deep deterministic policy gradient (DDPG) [14], an actor-critic, model-free DRL method. The agent of DDPG contains two DNNs: actor A and critic C with parameters $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_C$. Actor decides the action with respect to $\mathbf{s}_t$, and critic calculates $Q(\mathbf{s}_t, \pi(\mathbf{s}_t|\boldsymbol{\theta}_A)|\boldsymbol{\theta}_C)$ to evaluate the performance of actor. Critic can be trained the same as DQN, and actor can be trained using policy gradient and chain rule with respect to expected cumulated reward J [14], namely, $\nabla_{\boldsymbol{\theta}_\pi} J = E[\nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}_C) \cdot \nabla_{\boldsymbol{\theta}_A} \pi(\mathbf{s}|\boldsymbol{\theta}_A)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi(\mathbf{s}|\boldsymbol{\theta}_A)}]$. Readers could refer to [15]–[20] for more variants of DRL.

C. DRL-Based TCP Congestion Control Method

In 2019, Xiao *et al.* proposed TCP-Drinc [21], which applies one DQL agent for each TCP flow to control its *cwnd* and achieves better performance compared with baselines. However, when there are many TCP flows, TCP-Drinc highly consumes computing resources. Besides, TCP-Drinc adopts the implicit assumption (i.e., the feedback of one data-sending rate controlling action returns at an RTT after the action is applied). Also in 2019, Xu *et al.* proposed DRL-CC [11], a congestion control method focusing on multi-path TCP (MPTCP). Despite the fact that DRL-CC achieves significant performance improvement on throughput over other baselines, it ignores the delayed feedback issue and the reward applied in DRL-CC only considers throughput, which might be harmful in some scenarios with low bottleneck bandwidth. Also, it can only adjust the *cwnd* of one flow, as addressed by Xu *et al.*, "at each epoch (i.e., time-step), DRL-CC only takes an action on one (target) MPTCP flow" [11], which makes it inefficient to deal with scenarios with many TCP/MPTCP flows. In 2020, Emara *et al.* proposed Eagle [22], which introduces expert experience in DRL-based congestion control. It takes TCP BBR as an expert to generate some experience to speedup its training process while enable exploration to find better action. In 2020, Cui *et al.* proposed Hd-TCP [23], which focus on congestion control in high-speed railway scenarios, where users' signal quality often degrades severely, and handover among base stations frequently happens. It has also proved that DRL can find better policy for congestion control in highly dynamic RAN scenarios. In 2020, Zhang *et al.* proposed ARC [24], which is a framework to train and deploy DRL-based congestion control methods in real network environment by asynchronous learning methods. It has also demonstrated that DRL-based congestion control can work in real network. In 2021, He *et al.* proposed DeepCC [25], which applies multi-agent DRL methods in MPTCP congestion control. It has also empowered DRL with self-attention mechanism to deal with changing number of MPTCP flows. Other similar methods include Rax [26], TCP-RL [27], SmartCC [28] etc.

[3] $Q(\mathbf{s}_t, \mathbf{a}_t)$ is the Q-value estimated in the implementation of reinforcement learning, and $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ is the accurate Q-value.

III. System Model and Problem Formulation

A. System Model

In this paper, an RAN-included network architecture is adopted, as shown in Fig. 1. A set of mobile users are connected to a base station (BS) through RAN. The BS is connected with the server through a core network (CN), which is multi-hop and multi-path. TCP flows are established between users and the server through the network and data is transmitted from the server to users. The path of each TCP flow is pre-decided. Note-worthily, DRL-3R should be deployed at the data-sender, i.e., server in this paper. In DRL-3R, RAN information is considered as a part of the state space to control *cwnd*. According to the general principle of congestion control, *cwnd* should be decreased when congestion happens due to sophisticated factors, such as extremely high sending rate and low-quality channel. Due to the highly dynamic and volatile wireless channels, RAN can be considered as a potential bottleneck in the whole network, which might significantly affect the utility of the whole network. However, with the advancement of wireless technologies, RAN should not always be treated as the bottleneck. To better leverage this useful auxiliary RAN information, in DRL-3R, we leverage the RAN information prediction module to timely learn the dynamic of a potential bottleneck, so as to become more vigilant when RAN (potentially) becomes a bottleneck. In particular, we choose the user-received signal strength and the physical resource block (PRB) available ratio as *the RAN state*, since the user-received signal strength decides the transmission rate of each user directly while PRB available ratio indicates the probability of each user getting PRBs. Specifically, the PRB available ratio of user $u \in \mathcal{U}$ is calculated as the proportion of user u getting needed PRBs in the latest several time-slots. Notably, though we assume that users can collect these information in real time and transmit them to the server piggy-backed by the ACK segment, the latency from user to server should not be neglected and will change dynamically due to the variations of wireless channel. Meanwhile, information of TL is taken into account in DRL-3R, consistent with [11], [21]. Specifically, we consider five kinds of TL information, including the change of *cwnd* and throughput in the past two consecutive time-steps, the arriving interval of ACK segment

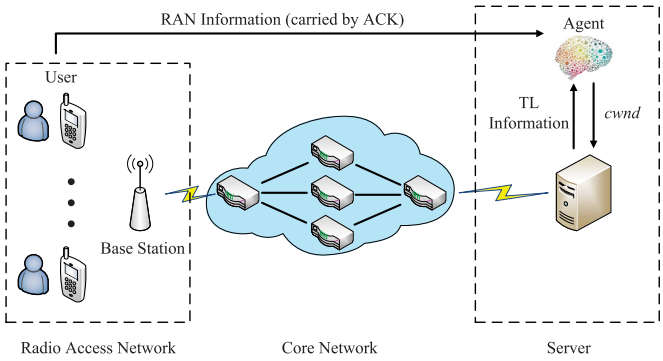


Fig. 1. A typical TCP scenario consisting of both RAN and CN, and applying DRL-3R at the server.

(except duplicate ACK segment), RTT, and the ratio of current minimum RTT over current RTT, which are referred as *the TL state*.

B. Problem Statement

As shown before, our goal is to adjust *cwnd* properly to achieve a superior performance. Similar to [21], we treat congestion control as a *dynamic-delayed, equilibrium-finding and fairness-needed decision-making problem*.

Firstly, it is a *dynamic-delayed problem*, which means that the feedback of each action would not return at a fixed time. Some researches (e.g. [21]) assume that an action should receive its feedback at an RTT after the action is applied. In other words, the feedback returning at time t is only decided by $\mathbf{a}_{t-RTT}$. However, this assumption may not hold in network as RTT is changing over time. As the server can't predict the real-time RTT accurately, it is not practical to make this assumption. Also, the feedback acquired by this assumption may contain feedback of several consecutive actions. In this paper, inspired by RUDDER [7], we propose similar reward redistribution to deal with this problem. Compared with the assumption noted before, DRL-3R relaxes this assumption into a weaker one.

Secondly, congestion control is an *equilibrium-finding problem*. Throughput and RTT are mutually exclusive. Therefore, we need to find an equilibrium in this case so that we can get a good throughput with an acceptable RTT, and DRL-3R is capable of achieving this goal.

In the end, congestion control is a *fairness-needed problem*. When a set of TCP flows share the limited bandwidth, it is necessary to maintain the fairness among all TCP flows. In this paper, the α-fairness function is adopted to ensure the trade-off between fairness and utilization. DRL-3R achieves a higher fairness compared with baselines.

C. Problem Formulation

Beforehand, the definition of time-step and time-slot should be clarified. Time-slot is the minimum time granularity considered in this paper, while time-step, an integral multiple of time-slots, is defined as the agent's decision-making frequency. For example, setting a time-step equal to 10 time-slots means that the agent makes a decision every 10 time-slots.

Inspired by [21], we define the long-term throughput $x(t)$ as

$$x(t) = \sum_{\kappa=t}^{t+L_{hor}-1} \hat{x}(\kappa) \cdot \eta^{\kappa-t} \tag{2}$$

where L_{hor} is the number of time-steps considered by $x(t)$ (i.e., horizon), η is the decay coefficient limited in range $(0, 1]$, and $\hat{x}(\kappa)$ is the throughput measured at time κ. Note that this equation calculates the long-term throughput of *a* TCP flow instead of *all* flows. We use this form as the action (i.e., change of *cwnd*) has long-tail effect, i.e., the impact of an action lasts for a period of time but decreases exponentially as [21] suggested.

Finally, we propose the utility function $U(t)$ as

$$U(t) = \sum_{u \in \mathcal{U}} \gamma_u \cdot U_{\alpha_t}(x_u(t)) - \sum_{u \in \mathcal{U}} \delta_u \cdot U_{\alpha_r}(RTT_u(t)) \tag{3}$$

where $x_u(t)$ is the long-term throughput of u, γ_u and δ_u are weights which define the relative importance of throughput and RTT, and α_t and α_r are fairness parameter of long-term throughput and RTT, respectively. In particular, we adopt the α-fairness function as [29]

$$U_\alpha(\kappa) = \begin{cases} \log(\kappa), & \text{if } \alpha = 1 \\ \dfrac{\kappa^{1-\alpha}}{1-\alpha}, & \text{else} \end{cases} \tag{4}$$

where α is the fairness parameter. For simplicity, we set all users' γ_u and δ_u are identical, and $\alpha_t = \alpha_r = \alpha$.

D. DRL-Based TCP Congestion Control

In this paper, we adopt DRL to jointly optimize the throughput and RTT. In this section, we elaborate the details of our MDP, i.e., state space $\mathcal{S}$, action space $\mathcal{A}$ and reward $\mathcal{R}$.

1) State: Our state space $\mathcal{S}$ contains two parts, i.e., RAN state and TL state of each TCP flow, whose details are shown in Section III-A. Notably, throughput and ACK arriving interval are processed by exponentially weighted moving-average (EWMA). The ratio of minimum RTT over RTT works as the ratio of propagation delay over total RTT, as the minimum RTT can be considered as the estimation of propagation delay. The minimum RTT can be defined as the minimum RTT among all flows at current time-step. Finally, the state at time-step t, i.e., $\mathbf{s}_t$, is a concatenation of RAN state and TL state, and contains states of all TCP flows.

2) Action: The action is set as the change of *cwnd* of each TCP flow. Let $\mathbf{a}_t = \{a_{u,t}, u \in \mathcal{U}\}$ be an $|\mathcal{U}|$-dimension action vector. The action can be applied as $\text{cwnd}_{u,t} = 2^{a_{u,t}} \cdot \text{cwnd}_{u,t-1}$. Note that each component of $\mathbf{a}_t$ is limited in $[-1, 1]$. The minimum and maximum *cwnd* of each TCP flow may also be limited based on the environment. This setting of actions to *cwnd* enables that the new *cwnd* can be twice as the last *cwnd* in maximum or half in minimum, which facilitates a flexible control and fast correction of *cwnd*. It also ensures that the maximum instantaneous changing rate of *cwnd* is consistent with TCP Reno's, to prevent over-adding or over-reducing of *cwnd*.

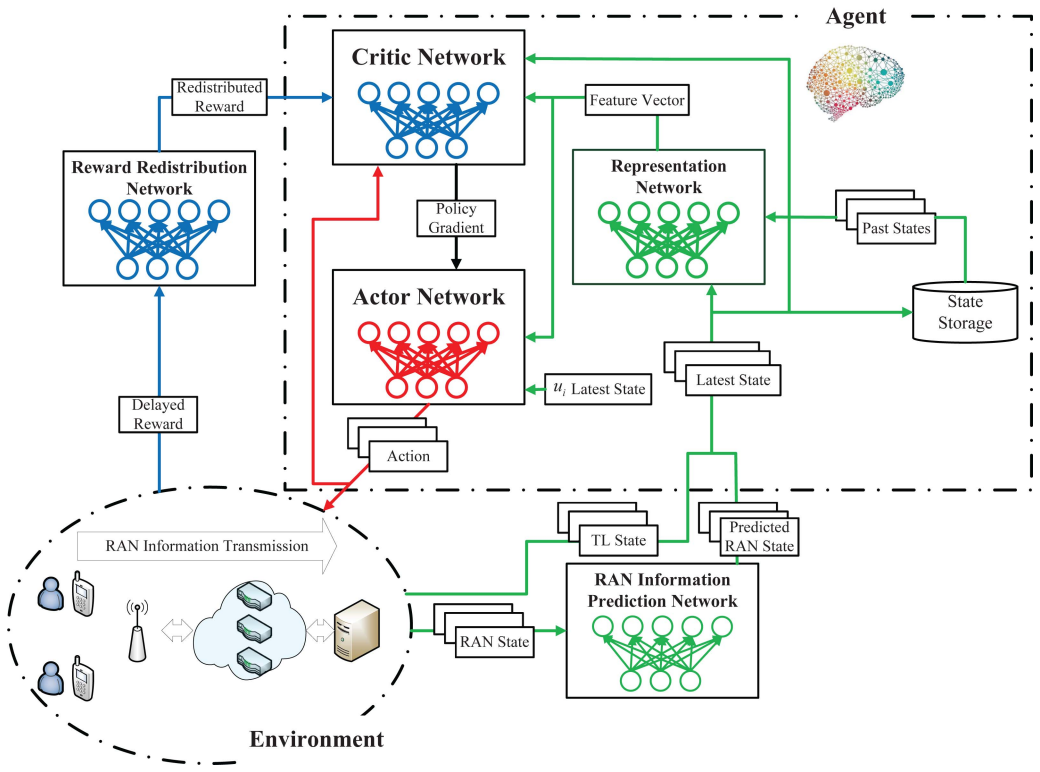


Fig. 2. An illustration of the architecture of DRL-3R. State and action represented by three blocks (e.g., predicted RAN State) indicate global state and action, while state represented by one block indicate local one. Target network and replay buffer are not shown in this figure.

3) Reward: We use the utility function defined in Section III-C to calculate the delayed reward. We cut the whole state-action sequence of an episode into several short sequences with equal length L, and calculate a delayed reward for each sequence. In the following Section IV-C, we will explain how to redistribute the delayed reward to each action in detail.

For a sequence starting at time-step t and ending at time-step $t+L-1$, its delayed reward is defined and calculated as $\hat{R}_{t+L} = U(t+2L-1) - U(t+L-1)$. Reward clipping is also used to limit the delayed reward in $[-1, 1]$. This delayed reward indicates that how much effect this sequence of actions have on utility function.

IV. Architecture and Designation of DRL-3R

A. Overview

As Fig. 2 shows, DRL-3R mainly encompasses three parts, i.e., RAN information prediction, reward redistribution, and representation network-enforced DDPG based DRL. At each time-step t, the environment provides RAN and TL state to the agent. We apply RAN information prediction to predict the latest RAN state. Afterwards, the global state (i.e., predicted RAN state and TL state of all TCP flows) is sent into the representation network, which extracts a feature vector from several latest state. Then, the feature vector is provided to actor and critic. In action selection process, the actor decides the action to separately control *cwnds* of all individual TCP flows, by optimizing the global utility with shared information (i.e., state) of all TCP flows connected to the same server. While deciding the action on TCP flow related with user u (i.e., $a_{u,t}$), the actor receives the local RAN and TL state (i.e., the RAN and TL state of u) and the feature vector. In the learning process, the environment provides delayed reward to reward redistribution module at a specific time-step. The reward redistribution module redistributes the delayed reward to each time-step and gets redistributed reward, which finally constitutes the experience (i.e., state, action, redistributed reward, next state) which can be learned directly by representation network-enforced DDPG. From a general perspective, DRL-3R is trained neither for a TCP flow nor a client or user. It is trained for all TCP flows connected to the same server. Also note that such settings are also consistent with DRL-CC [11], and we only consider one server in this paper.

B. RAN Information Prediction

The RAN information prediction module provides the estimated RAN state to the agent. We assume that users can get RAN state in real time, and transmit it to the server with ACK segment. At each time-step, the agent at the server gets a sequence of RAN state with exact collected timestamp to predict the latest RAN state. We simply use two long short-term memory (LSTM)-based neural networks with the same structure to predict user-received signal strength and PRB available ratio separately. Moreover, since RAN state may not be collected in constant time interval, we include the time

interval of RAN state in the prediction as a feature. The label (i.e. true value of RAN state) can be easily collected as RAN state can be acquired from BS or user at each time-slot. Therefore, we can set up two datasets to train two prediction networks separately.

C. Reward Redistribution

As explained in Section III-B, the congestion control problem is a dynamic-delayed problem, which means that the feedback of each action would not return at a fixed time interval. Traditionally, it is widely assumed that the feedback induced by action $\mathbf{a}_t$ taken at time t returns at $t + RTT$. However, this assumption may not hold as the server can't predict real-time RTT accurately. Meanwhile, feedback could be induced by several consecutive actions as well.

In this paper, inspired by RUDDER [7], we adopt reward redistribution to partly relax this assumption and compute redistributed reward from delayed ones. We define a delayed-rewarded MDP $\hat{\mathcal{M}}$ with an episode of $T+1$ time-steps starting from 0 to T, and the environment returns a delayed reward $\hat{R}_{T+1}$ at the end of the episode. The state-action sequence is defined as $(\mathbf{s}_0, \mathbf{a}_0), \cdots, (\mathbf{s}_T, \mathbf{a}_T)$.

To redistribute the delayed reward to each state-action pair and get redistributed rewards, we firstly introduce the concept of *sequence-Markov decision process* (SDP), which is quite similar to MDP. The only difference between SDP and MDP is that the reward of SDP could be not Markov, while the reward of MDP must be Markov. More precisely, the reward in MDP is only decided by the former state and action. Instead, the reward in SDP *may* be decided by several consecutive state-action pairs, which allows the existence of delayed reward, because delayed reward is decided by several state-action pairs instead of only the former one. The definitions of other components of SDP and MDP, including state space, action space and state-transmission probability, remain identical. Clearly, $\hat{\mathcal{M}}$ is also an SDP.

Secondly, we introduce the concept of *second-order Markov reward redistribution*, which can redistribute the delayed reward and change an SDP with delayed reward into an SDP without delayed reward. For SDP $\hat{\mathcal{M}}$, a new SDP $\mathcal{M}$ without delayed reward, which shares the same state space, action space and state-transmission probability with $\hat{\mathcal{M}}$ but different reward, can be obtained by second-order Markov reward redistribution, which is defined as follows.

$$E[R_{t+1}|(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)] = \hat{Q}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}^{\pi}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \tag{5}$$

where $\hat{Q}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ is the Q-value of $(\mathbf{s}_t, \mathbf{a}_t)$ of SDP $\hat{\mathcal{M}}$, and R_{t+1} is the redistributed reward of $(\mathbf{s}_t, \mathbf{a}_t)$ of SDP $\mathcal{M}$. [7] proves that $\hat{\mathcal{M}}$ and $\mathcal{M}$ share the same optimal policy, so we can get the optimal policy of $\hat{\mathcal{M}}$ by solving $\mathcal{M}$. Besides, if the reward redistribution is optimal, i.e., R_{t+1} is independent of state-action pairs between $t+1$ and the end of episode, the estimation of the Q-value of $(\mathbf{s}_t, \mathbf{a}_t)$ of SDP $\mathcal{M}$ [i.e., $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$] can be simplified into the estimation of R_{t+1} while solving $\mathcal{M}$. Based on these ideas, we have three means to solve $\mathcal{M}$: (1) Q-value estimation, which estimates the $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ by estimating R_{t+1}. (2) Policy-gradient method, which replaces $Q(\mathbf{s}_t, \mathbf{a}_t)$ with R_{t+1}, as both R_{t+1} and $Q(\mathbf{s}_t, \mathbf{a}_t)$ can be seen as the estimation of $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$. (3) Q-learning, which treats $\mathcal{M}$ as an MDP.

To realize the second-order reward redistribution, we define a *return-prediction function* g to predict the expected cumulative reward (i.e., return) of $\hat{\mathcal{M}}$ for a given state-action sequence ending at time-step t, i.e., $\hat{Q}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$, and the output of g is considered as $\hat{Q}(\mathbf{s}_t, \mathbf{a}_t)$. We also define a *difference function* $\Delta[(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)]$, to calculate the specific information contained by $(\mathbf{s}_t, \mathbf{a}_t)$, because $(\mathbf{s}_t, \mathbf{a}_t)$ should contain the information of $(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$ according to Markov property. Δ can be defined according to the environment. In this paper, we simply define Δ as the numerical difference between state-action pairs $(\mathbf{s}_t, \mathbf{a}_t)$ and $(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$. The state-action sequence processed by the function Δ can be rewritten as follows.

$$\begin{aligned}&\Delta_{0:T}\\ &= \{\Delta[(\mathbf{s}_{-1}, \mathbf{a}_{-1}), (\mathbf{s}_0, \mathbf{a}_0)], \cdots, \Delta[(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}), (\mathbf{s}_T, \mathbf{a}_T)]\}\end{aligned} \tag{6}$$

Then, g should ensure

$$g(\Delta_{0:T}) = \hat{Q}(\mathbf{s}_T, \mathbf{a}_T) = \hat{R}_{T+1} \tag{7}$$

This equation can be explained as follows. On the one hand, $\hat{Q}(\mathbf{s}_T, \mathbf{a}_T)$ is calculated by $g(\Delta_{0:T})$ and works as an estimation of $\hat{Q}^{\pi}(\mathbf{s}_T, \mathbf{a}_T)$. On the other hand, $\hat{Q}(\mathbf{s}_T, \mathbf{a}_T) = \hat{R}_{T+1}$ should be ensured because $\hat{Q}^{\pi}(\mathbf{s}_T, \mathbf{a}_T)$ is the expected cumulative reward after time-step T, and the only reward after T is $\hat{R}_{T+1}$. If a neural network is applied as g, this equation defines the label during its training process. Proposed by [7], $g(\Delta_{0:T})$ can be decomposed into

$$g(\Delta_{0:T}) = \sum_{t=0}^{T} h_t \tag{8}$$

where h_t is the contribution of $(\mathbf{s}_t, \mathbf{a}_t)$ to the predicted expected return. Similarly, $g(\Delta_{0:t})$ can also be decomposed into

$$g(\Delta_{0:t}) = \sum_{i=0}^{t} h_i \tag{9}$$

Clearly, h_t can be calculated by $g(\Delta_{0:t}) - g(\Delta_{0:t-1})$. Specially, we set $h_0 = g(\Delta_{0:0})$. Note-worthily, the relationship between h_t and R_{t+1} can be described as

$$E[R_{t+1}|(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)] = h_t \tag{10}$$

We leave the proof of this equation in Appendix A. With respect to the function g, there could be an error between $g(\Delta_{0:T})$ and $\hat{R}_{T+1}$. Therefore, an extra redistributed reward R_{T+2} is added.

$$R_{T+2} = \hat{R}_{T+1} - g(\Delta_{0:T}) \tag{11}$$

In the end, we conclude the basic calculation method of reward redistribution as

$$\begin{aligned}E[R_1|(\mathbf{s}_0, \mathbf{a}_0)] &= h_0 = g(\Delta_{0:0})\\ E[R_{t+1}|(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)] &= h_t = g(\Delta_{0:t}) - g(\Delta_{0:t-1})\end{aligned} \tag{12}$$

Algorithm 1 Reward Redistribution

Input: State-action sequence $(\mathbf{s}_0, \mathbf{a}_0), \cdots, (\mathbf{s}_T, \mathbf{a}_T)$, return-prediction function g, difference function Δ, $0 \leq t \leq T$
Output: Expectation of redistributed reward h_t
1: Calculate $\Delta[(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)]$;
2: Calculate $g(\Delta_{0:t}) = g\{\Delta[(\mathbf{s}_{-1}, \mathbf{a}_{-1}), (\mathbf{s}_0, \mathbf{a}_0)], \cdots, \Delta[(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)]\}$;
3: Calculate h_t and R_{T+2} by Equ. (12) and Equ. (11);
4: **return** h_t, R_{T+2}

Algorithm 1 shows the complete process of reward redistribution and Fig. 3 illustrates this algorithm from another perspective. For a state action pair $(\mathbf{s}_t, \mathbf{a}_t)$, before it is executed, the expected return is $g(\Delta_{0:t-1})$. Once executed, the expected return is $g(\Delta_{0:t})$. The numerical difference between $g(\Delta_{0:t})$ and $g(\Delta_{0:t-1})$ (i.e., h_t) can be seen as the contribution of $(\mathbf{s}_t, \mathbf{a}_t)$. Moreover, an LSTM-based network can be used to predict g, as g needs to accept length-varying input. During the training process, $\Delta_{0:T}$ is used as input while $\hat{R}_{T+1}$ is treated as the label. Finally, $\hat{R}_{T+1}$ has been redistributed to each time-step, which finally constitutes a new SDP $\mathcal{M}$ and can be solved with traditional RL methods.

Furthermore, we make the following modifications on the aforementioned reward redistribution to better fit our environment. (1) We use the mathematical expectation of R_{t+1} as its true value, i.e., $h_t = R_{t+1}$. (2) In our environment, if we only return a final delayed reward at the end of each episode (which is a whole *cwnd*-controlling process lasting for several seconds), it would be very difficult to redistribute it to every time-step, because the RAN-included network is time-varying in a fast manner, and the final delayed reward can't contain the dynamic information during an episode. Hence, we cut each episode into several shorter sequences with length L, return a reward for each sequence separately as described in Section III-D, and redistribute it to each action. (3) As we could not add an extra reward R_{T+2} at the end of each sequence, we apply a new method called *redistribution correction*, which is shown in Algorithm 2 to ensure Equ. (7). Finally, we propose the following theorem on our reward redistribution.

Theorem 1: The reward redistribution applied by DRL-3R ensures that $\hat{\mathcal{M}}$ shares the same optimal policy with $\mathcal{M}$ constituted by the same state space, the same action space and redistributed reward.

We leave the proof in Appendix B.

Algorithm 2 Redistribution Correction

Input: Return-prediction function $g(\Delta_{0:t}), 0 \leq t \leq T$, and delayed reward $\hat{R}_{T+1}$
Output: Redistributed reward R_{t+1}
1: Calculate $R_{T+2} = \hat{R}_{T+1} - g(\Delta_{0:T})$;
2: Calculate uncorrected redistributed reward $R'_{t+1} = g(\Delta_{0:t}) - g(\Delta_{0:t-1})$ for $t \neq 0$, and $R'_1 = g(\Delta_{0:0})$;
3: Calculate the average error $E = \frac{R_{T+2}}{T+1}$;
4: Correct redistributed reward $R_{t+1} = R'_{t+1} + E$;
5: **return** R_{t+1}

D. Representation Network

Inspired by DRL-CC [11], we also apply the representation network as a feature extractor to extract the time-series relationship among global state from several latest time-steps. As global state contains the state of all TCP flows at each time-step, the extracted feature also contains information from all TCP flows, and can be considered as a more complete global feature compared with DRL-CC. We denote the maximum number of time-steps considered by representation network as L_{rep}. The global state sequence $\mathbf{s}_{\text{seq},t} = \{\mathbf{s}_{t-L_{\text{rep}}+1}, \mathbf{s}_{t-L_{\text{rep}}+2}, \cdots, \mathbf{s}_t\}$ works as input to representation network, and the output vector works as the feature vector which contains global information. Similar to DRL-CC, the representation network of DRL-3R can also be trained together with actor and critic in an end-to-end manner. Xu *et al.* [11] believe that this could bring better performance than training separately.

E. Representation Network-Enforced DDPG-Based DRL

In this section, we formally introduce the core part of DRL-3R, which is based on DDPG [14]. Note that in Section I, we address that DRL-3R tries to relax the implicit assumption into a weaker one. With the reward redistribution described in Section IV-C and delayed reward described in Section III-D, our assumption can be addressed as follows.

Assumption 1: The feedback of actions in each sequence would return at *around* L time-steps after the end of the sequence.

The characteristic of Assumption 1 is as follows. (1) We do not assume that the feedback returns *at* or *before* a specific time-step. Instead, we allow the feedback returning *around* a specific time-step. If the feedback of an action returns shortly before L time-steps after the end of the sequence (i.e., time-step $t + 2L - 1$), its feedback would be considered by the calculation of utility function at time-step $t + 2L - 1$. On the contrary, its feedback would also be considered according to the definition of utility function in Equ. (3). (2) We acknowledge that though we can't calculate the feedback or reward of each action accurately, we directly apply the delayed reward reflecting the influence of a series of actions and try to redistribute the delayed reward to each action.

On top of the DDPG introduced in Section II-B, we integrate it with the modules (i.e., the RAN information prediction module and the reward redistribution module) and formally propose the process of DRL-3R, as shown in Algorithm 3.

V. Performance Evaluation

A. Experiment Setup

In our simulation scenario, there are 30 users located in a square with side length of 80 meters surrounding the BS located at the center. All users are moving with the speed of $2m/s$ and a fixed direction angle θ. The direction and location of each user are initialized randomly at the beginning of each episode. As shown in Fig. 1, the BS is connected with CN via one edge router. Furthermore, there only exists one link between the BS and the edge router. 3 independent identical links bridge edge router and the server with another router

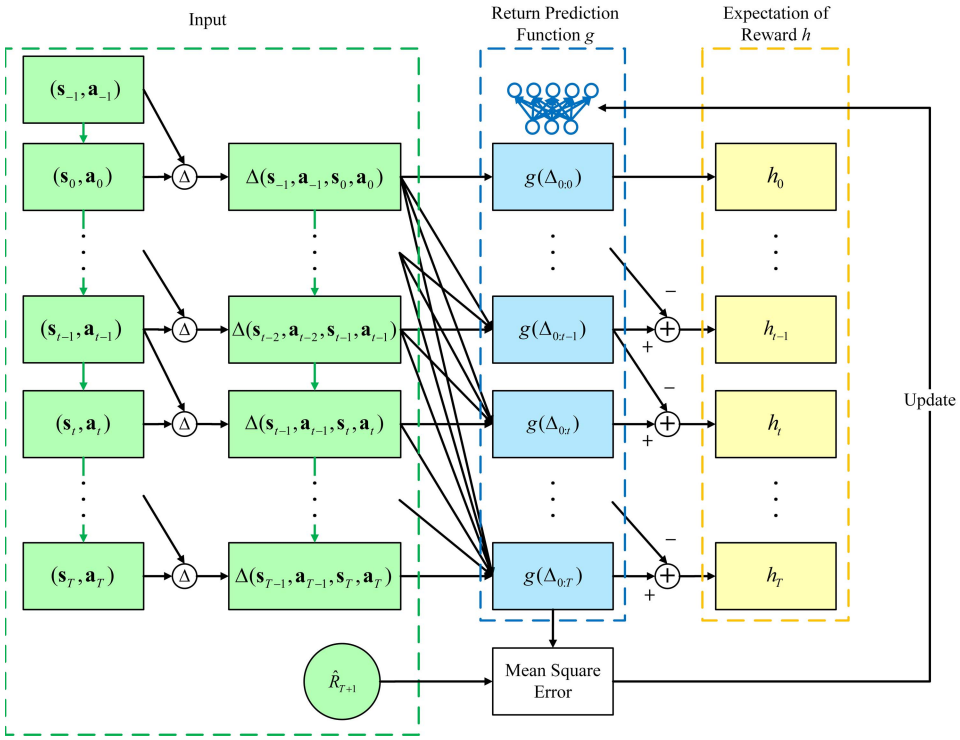


Fig. 3. An illustration of the implementation method of reward redistribution. g is the return-prediction function which accepts varying-length sequence, Δ is the difference function. Green solid arrays means state-action transitions between different time-steps, black solid arrays means neural network training and calculation process of reward redistribution.

on each link. We set that each link is shared by 10 TCP flows, and the path of each TCP flow remains identical in our whole experiment. We assume a full-buffer server, which always has data to transmit and sends in a 'bulky' manner, i.e., its application tries to send data as fast as possible, and the maximum sending rate of each TCP flow is limited as 23.4 Mbps to simulate the phenomenon where the server aims to guarantee service provisioning to more users by limiting the maximum sending rate of each TCP flow. However, it should be noted that this maximum sending rate is 6 times of the biggest downlink bottleneck bandwidth, and the competition among TCP flows can be effectively simulated. The length of each episode is set as 2 seconds to simulate scenarios with relatively shorter flows in Internet (e.g., downloading short video). The length of a time-slot is set as 0.5 milliseconds and the length of a time-step is set as 20 milliseconds (40 time-slots). The minimum *cwnd* of DRL-3R is set as 1 maximum segment size (MSS) and the maximum is set as 50 MSSs.

We build the aforementioned environment using Python and DRL-3R using Python and Pytorch [30], and use DRL-3R to replace traditional congestion control methods, so as to "take over" the control of *cwnd* from traditional methods. Currently, we don't consider the connection between transport layer and application layer,[4] and assume that the program of DRL-3R can directly control cwnds of all TCP flows and get needed information, i.e., RAN state and TL state. Besides, basic functions of TCP protocol remain unchanged. We test DRL-3R in several typical scenarios, including three steady scenarios (Scenario 1-3) and three dynamic scenarios (Scenario 4-6). We compare DRL-3R with a few representative congestion control methods, including TCP Reno [2], TCP Westwood [8], TCP Cubic [9], TCP BBR [10] and DRL-CC [11]. For TCP BBR, when the estimated bottleneck bandwidth's improvement of two rounds is less than 25%, it would change from start phase to drain phase, to prevent over-estimation of the bandwidth. For DRL-CC, the goodput in its state space and derivation of RTT are defined the same as throughput in this paper, and devRTT in TCP protocol, respectively. To ensure fair comparison between DRL-3R and DRL-CC, DRL-CC shares the same maximum *cwnd* changing rate, architecture of actor, critic and representation networks, and other key RL parameters (e.g., time-step, replay buffer size, etc.) as DRL-3R. Furthermore, the output of DRL-CC's actor is limited in $[-1, 1]$ and is transformed linearly to control the *cwnd*.

B. Pre-Training

In DRL-3R, the neural network of RAN information prediction module and reward redistribution module need to be pre-trained. We firstly generate datasets to train RAN information prediction networks. For each kind of RAN information (i.e., user-received signal strength and PRB available ratio), a specific dataset, which is leveraged to train an independent neural network, is generated by running the simulation scenario for

[4]We have testified the feasibility to connect a more realistic platform (e.g., ns-3) with Python and Pytorch inspired by ns3-gym [31]. However, due to the space limitation, we leave the more comprehensive and careful evaluation as future works.

Algorithm 3 DRL-3R for TCP Congestion Control

Input: noise θ_n, batch size N_{batch}, replay buffer size N_{RB} and soft update parameter τ;
/* Initialization. */
Initialize representation network R, actor network π, critic network Q with random parameters $\boldsymbol{\theta}_R$, $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_Q$, respectively, and target network R', π', Q' with parameters $\boldsymbol{\theta}_{R'} = \boldsymbol{\theta}_R$, $\boldsymbol{\theta}_{\pi'} = \boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_{Q'} = \boldsymbol{\theta}_Q$, respectively;
Initialize replay buffer $\mathbf{RB}$ with N_{RB};
Initialize noise with θ_n;
/* Action selection and experience processing. */
for every episode **do**
  for every time-step **do**
    Acquire RAN state sequences for all users, and predict the latest RAN state, and acquire TL state of all users, and get global state $\mathbf{s}_t$;
    Derive feature vector $\mathbf{f}_t$ from L_{rep} latest states by R: $\mathbf{f}_t = R(\mathbf{s}_{t-L_{\text{rep}}+1}, \cdots, \mathbf{s}_t|\boldsymbol{\theta}_R) = R(\mathbf{s}_{\text{seq},t}|\boldsymbol{\theta}_R)$;
    Derive an action $\mathbf{a}_t$ by π: $\mathbf{a}_t = \{a_{u,t} = \pi(\mathbf{e}_{u,t}, \mathbf{f}_t|\boldsymbol{\theta}_\pi), u \in \mathcal{U}\}$, where $\mathbf{e}_{u,t}$ is the local state of user u at time-step t, and acquire a sample from noise, add it to $\mathbf{a}_t$, and execute $\mathbf{a}_t$;
    Store $\mathbf{s}_{\text{seq},t}$, $\mathbf{a}_t$ and current utility data (i.e., throughput and RTT of all flows);
  end for
  Calculate delayed reward for each sequence according to utility data, and redistribute delayed reward to each time-step t as redistributed reward R_{t+1} by Algorithm 1 and 2;
  Store transition $\{\mathbf{s}_{\text{seq},t}, \mathbf{a}_t, R_{t+1}, \mathbf{s}_{\text{seq},t+1}\}$ in $\mathbf{RB}$;
  /* Learning. */
  Sample N_{batch} transitions $\{\mathbf{s}_{\text{seq},j}, \mathbf{a}_j, R_{j+1}, \mathbf{s}_{\text{seq},j+1}\}$ from $\mathbf{RB}$, $j \in 1, 2, \ldots, N_{\text{batch}}$;
  Derive feature vector $\mathbf{f}_{j+1}$ by R': $\mathbf{f}_{j+1} = R'(\mathbf{s}_{\text{seq},j+1}|\boldsymbol{\theta}_{R'})$;
  Decide next action $\mathbf{a}_{j+1}$ by π': $\mathbf{a}_{j+1} = \{a_{u,j+1} = \pi'(\mathbf{e}_{u,j+1}, \mathbf{f}_{j+1}|\boldsymbol{\theta}_{\pi'}), u \in \mathcal{U}\}$, where $\mathbf{e}_{u,j+1}$ is the local state of u at time-step $j+1$;
  Calculate the target value for critic by Q': $y_j = R_{j+1} + \gamma \cdot Q'(\mathbf{s}_{j+1}, \mathbf{f}_{j+1}, \mathbf{a}_{j+1}|\boldsymbol{\theta}_{Q'})$;
  Update the parameters of critic by minimizing the MSE loss between y_j and $Q(\mathbf{s_j}, R(\mathbf{s}_{\text{seq},j}|\boldsymbol{\theta}_R), \mathbf{a}_j|\boldsymbol{\theta}_Q)$;
  Update the parameters of actor network by policy gradients: $\frac{1}{N_{\text{batch}}} \sum_{j=1}^{N_{\text{batch}}} \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{f}, \mathbf{a}|\boldsymbol{\theta}_Q) \cdot \sum_{u \in \mathcal{U}} \nabla_{\boldsymbol{\theta}_\pi} \pi(\mathbf{e}_u, \mathbf{f}|\boldsymbol{\theta}_\pi)$, for $\mathbf{s} = \mathbf{s}_j$, $\mathbf{f} = R(\mathbf{s}_{\text{seq},j}|\boldsymbol{\theta}_R)$, $\mathbf{a} = \{a_{u,j}\}$, $a_{u,j} = \pi(\mathbf{e}_u, \mathbf{f}|\boldsymbol{\theta}_\pi)$, $\mathbf{e}_u = \mathbf{e}_{u,j}$;
  Update the parameters of representation network by policy gradients: $\frac{1}{N_{\text{batch}}} \sum_{j=1}^{N_{\text{batch}}} \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{f}, \mathbf{a}|\boldsymbol{\theta}_Q) \cdot \sum_{u \in \mathcal{U}} \nabla_{\mathbf{f}} \pi(\mathbf{e}_u, \mathbf{f}|\boldsymbol{\theta}_\pi) \cdot \nabla_{\boldsymbol{\theta}_R} R(\mathbf{s}_{seq}|\boldsymbol{\theta}_R)$, for $\mathbf{s} = \mathbf{s}_j$, $\mathbf{f} = R(\mathbf{s}_{\text{seq},j}|\boldsymbol{\theta}_R)$, $\mathbf{a} = \{a_{u,j}\}$, $a_{u,j} = \boldsymbol{\pi}(\mathbf{e}_u, \mathbf{f}|\boldsymbol{\theta}_\pi)$, $\mathbf{e}_u = \mathbf{e}_{u,j}$;
  Update the parameters of representation network, actor and critic with soft update: $\boldsymbol{\theta}_{Q'} = \tau\boldsymbol{\theta}_Q + (1-\tau)\boldsymbol{\theta}_{Q'}$, $\boldsymbol{\theta}_{R'} = \tau\boldsymbol{\theta}_R + (1-\tau)\boldsymbol{\theta}_{R'}$, $\boldsymbol{\theta}_{\pi'} = \tau\boldsymbol{\theta}_\pi + (1-\tau)\boldsymbol{\theta}_{\pi'}$;
end for

many episodes and taking random action sampled from a uniform distribution in the range of $[-1, 1]$ at each time-step since we have no prior knowledge of the distribution of agent's action. Therefore, we can get RAN information sequences and their label (i.e., the true value) at each time-step. In our experiment, each dataset consists about 0.2 million samples and MSE is used to train RAN information prediction networks. Then, based on pre-trained RAN information prediction networks, another dataset is generated to train reward redistribution network. During the generation, the same methodology is applied to generate about 0.18 million samples with state-action sequence and its label (i.e. delayed reward). Notably, both RAN information prediction networks and reward redistribution network can converge in hundreds of epochs.

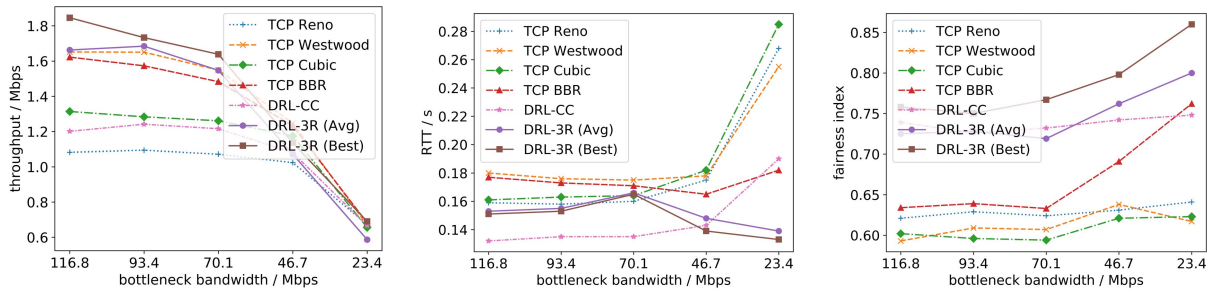
C. System Implementation

In this part, we consider some details of the implementation of DRL-3R. Firstly, to realize the function of DRL-3R, we need to modify the core codes of clients or users, so as to acquire RAN information by piggybacking it in the header of TCP segments. A flag bit should be added in the segments of three-way handshake, to notify data-sender whether RAN information is supported. Secondly, we emphasize that the overhead caused by DRL-3R is quite acceptable. We introduce extra overhead in two aspects, *transmission overhead* and *calculation overhead*. Transmission overhead is caused by piggybacking RAN information in ACK segment. In our experiment, 4 floats, which indicate user-received signal strength, PRB available ratio, and exact timestamp when they are collected respectively, need to be piggybacked in each ACK segment. Specifically, assuming each ACK segment can also carry useful data, and the maximum transmission unit (MTU) is 1500 bytes, it takes 16 bytes to piggyback RAN information, causing only 1.1% overhead and can be accepted. Calculation overhead is caused by the forward calculation of neural network. With an open-source tool "ptflops" [32], we find that DRL-3R only takes 659.25k multiply-accumulate (MAC) in each decision process in our experiment, which is quite small for mainstream graphics or central processing unit (GPU, CPU).

D. Test Scenarios and Results

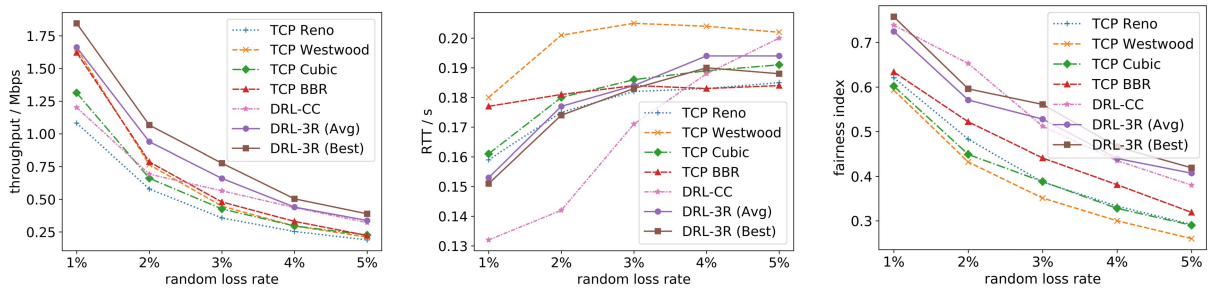
In this part, we present key performance indicators, including average per-flow throughput, RTT and fairness index, to verify the performance of DRL-3R in several representative environments.

1) Scenario 1: In this scenario, we show how the bottleneck bandwidth affects the performance of DRL-3R. We set the random loss rate as 1% and the propagation delay of CN as 50 ms. Note that the random loss rate is the same for both CN and RAN, both downlink and uplink. The corresponding results are shown in Fig. 4. It can be observed that when the bottleneck bandwidth is relatively high, DRL-3R can achieve a high throughput while maintaining a lower RTT. With the decrease of bottleneck bandwidth, DRL-3R maintains a reasonable throughput, while significantly reducing RTT. Specifically, when the bottleneck bandwidth is 93.4 Mbps, DRL-3R achieves 54%, 2%, 31%, 7% and 36% improvement on throughput over TCP Reno, TCP Westwood, TCP Cubic, TCP BBR and DRL-CC respectively, and also leads about 20 ms reduction to the RTT compared with TCP Westwood and TCP BBR. When the bottleneck bandwidth is 23.4 Mbps, although DRL-3R only maintains similar throughput compared with baselines, it achieves superior performance on RTT, reducing 129 ms, 116 ms, 146 ms, 43 ms and 51 ms on RTT over TCP Reno, TCP Westwood, TCP Cubic, TCP



(a) Average per-flow throughput. (b) Average RTT. (c) Average fairness index.

Fig. 4. Performance of DRL-3R and baselines in Scenario with steady network (Scenario 1), in which bottleneck bandwidth changes, and random loss rate and CN propagation delay remain constant. "DRL-3R (Avg)" is the average performance of DRL-3R, and "DRL-3R (Best)" is DRL-3R's best performance in our experiment.



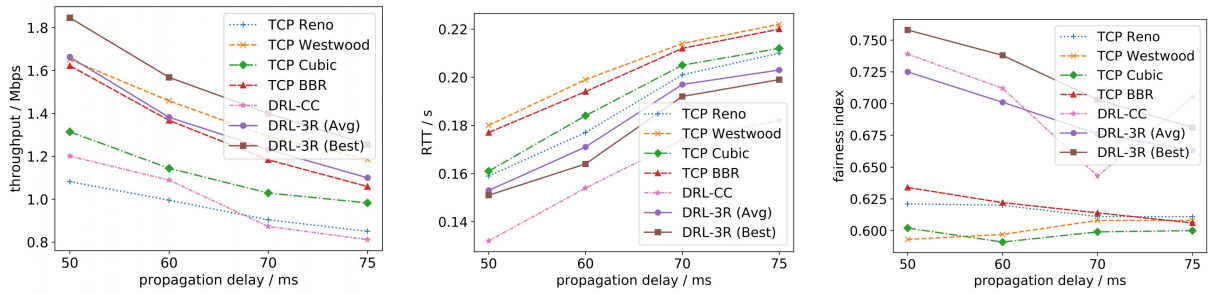
(a) Average per-flow throughput. (b) Average RTT. (c) Average fairness index.

Fig. 5. Performance of DRL-3R and baselines in Scenario with steady network (Scenario 2), in which random loss rate changes, and bottleneck bandwidth and CN propagation delay remain constant. "DRL-3R (Avg)" is the average performance of DRL-3R, and "DRL-3R (Best)" is DRL-3R's best performance in our experiment.

BBR and DRL-CC respectively. In summary, DRL-3R significantly outperforms other baselines. Besides, we also calculate the fairness index by calculating Jain's Fairness Index [33] of throughput at each time-step and averaging on all time-steps. It can be observed that DRL-3R achieves a higher fairness index among all cases. In this scenario, we retrain the RAN information prediction network and reward redistribution network in every case, and cut the weight of throughput in half when the downlink bottleneck bandwidth is relatively low to better fit the environment.

It can be found that DRL-CC can also outperforms some baselines. However, its lower RTT comes at a cost of lower throughput when the bottleneck bandwidth is relatively higher. This is because DRL-CC can only adjust the *cwnd* of one TCP flow at each time-step, which makes it inefficient on improving *cwnd* when a large amount of TCP flows exist. On the contrary, DRL-3R decides the action on each TCP flow in a round-robin manner, i.e., it decides actions of many TCP flows one by one, which enables DRL-3R to control the *cwnd* of many TCP flows in each time step, and increases the controlling efficiency of DRL-3R. Furthermore, when the bottleneck bandwidth decrease to 23.4 Mbps, another shortcoming of DRL-CC emerges that it tries to increase throughput while ignoring RTT. On the contrary, DRL-3R learns a smarter policy, maintaining a proper *cwnd*, as increasing the *cwnd* too much contributes trivially but leads to higher RTT. Comparing the performance of DRL-3R and DRL-CC, DRL-3R achieves a much better network utilization, achieving a significantly higher throughput when the bandwidth is high. In spite of a slight increase in RTT, we argue that DRL-3R still gets an RTT level similar to TCP Reno's. In other words, this increase can be seen as an acceptable cost to increase throughput. Therefore, we argue that DRL-3R achieves superior performance.

2) Scenario 2: In this scenario, we show how the random loss rate affects the performance of DRL-3R. We set the bottleneck bandwidth as 116.8 Mbps and the propagation delay of CN as 50 ms. Note that we pre-train the RAN information prediction network and reward redistribution network with bottleneck bandwidth as 116.8 Mbps, CN propagation delay as 50 ms and random loss rate as 1%. We don't retrain the pre-trained networks for each random loss rate, since the increase of random loss rate is often caused by unpredictable severe faults in practical network. Thus, we need the same group of pre-trained networks to deal with a series of possible random loss rate. The corresponding results are shown in Fig 5. It demonstrates that when the random loss rate is relatively small (i.e., 1%), the throughput of DRL-3R is similar to TCP Westwood's and BBR's, but DRL-3R achieves a higher fairness index and significantly smaller RTT. With the increase of random loss rate, DRL-3R significantly increases the throughput while maintaining a reasonable RTT. Specifically, when the random loss rate is 1%, DRL-3R leads to 54%, 27%, and 38% improvement on throughput over TCP Reno, TCP Cubic and DRL-CC respectively. Although DRL-3R doesn't achieve significant throughput improvement compared with TCP Westwood and TCP BBR, it saves about 25 ms on



(a) Average per-flow throughput. (b) Average RTT. (c) Average fairness index.

Fig. 6. Performance of DRL-3R and baselines in Scenario with steady network (Scenario 3), in which CN propagation delay changes, and bottleneck bandwidth and random loss rate remain constant. "DRL-3R (Avg)" is the average performance of DRL-3R, and "DRL-3R (Best)" is DRL-3R's best performance in our experiment.

RTT. When the random loss rate is 5%, DRL-3R achieves a much better performance, leading to 79%, 60%, 49%, 52% and 5% improvement on throughput over TCP Reno, TCP Westwood, TCP Cubic, TCP BBR and DRL-CC respectively, with an RTT similar to TCP Cubic. Note that DRL-3R always get better performance in terms of fairness index compared with most baselines. It can be also observed that the performance of DRL-CC is not as good as DRL-3R in most cases. DRL-CC's fairness is competitive with DRL-3R's, but its worse performance makes its reasonable fairness meaningless. Besides, DRL-3R is more robust to the change of random loss rate. Meanwhile, it can be observed that fairness index decreases quickly with the increasing of random loss rate. When packet loss happens in a TCP flow, the throughput of this TCP flow would decrease significantly before it turns stable. With the increase of random loss rate, increasingly more TCP flows' throughput would decrease while the others remain high, making fairness index decrease. In conclusion, DRL-3R achieves a superior performance on throughput and RTT while maintaining a better fairness compared with most baselines.

3) Scenario 3: In this scenario, we show how the propagation delay of CN affects the performance of DRL-3R. We set the random loss rate as 1% and the bottleneck bandwidth of CN as 116.8 Mbps. Moreover, we retrain the pre-trained neural networks while changing the propagation delay of CN. The corresponding results are shown in Fig. 6. We can see that DRL-3R achieves a high throughput and much lower RTT compared with baselines. Specifically, when the propagation delay of CN is set as 60 ms, DRL-3R achieves 39% and 21% improvement on throughput compared with TCP Reno and TCP Cubic respectively, while maintaining a similar throughput with TCP BBR and reducing 6 ms, 28 ms, 13 ms and 23 ms on RTT compared with TCP Reno, TCP Westwood, TCP Cubic, TCP BBR respectively. When the propagation delay of CN is set as 75 ms, DRL-3R achieves 29%, 12%, 4% and 36% improvement on throughput compared with TCP Reno, TCP Cubic, TCP BBR and DRL-CC respectively. It also reduce 7 ms, 19 ms, 9 ms and 17 ms on RTT compared with TCP Reno, TCP Westwood, TCP Cubic, TCP BBR respectively. This scenario shows that with the increasing of CN propagation delay, DRL-3R also achieves higher throughput while significantly reducing RTT, which implies that DRL-3R outperforms DRL-CC. Furthermore, DRL-3R achieves a better fairness compared with baselines in all cases.

4) Scenario 4: In this scenario, we present the performance of DRL-3R in a network with dynamic bottleneck bandwidth. We set the propagation delay as 50 ms and the random loss rate as 1%. We assume the bottleneck bandwidth is randomly chosen in 116.8 Mbps, 93.4 Mbps and 70.1 Mbps, with changing frequency as 10 Hz. The corresponding results are shown in Table I. It can be observed that the performance gain of DRL-3R is quite similar to that in Scenario 1. Specifically, DRL-3R achieves higher throughput compared with TCP Reno, TCP Westwood, TCP Cubic, TCP BBR and DRL-CC, and maintains a reasonable RTT. It also achieves a better fairness compared with baselines.

5) Scenario 5: In this scenario, we present the performance of DRL-3R in a network with dynamic loss rate. We set the bottleneck bandwidth as 116.8 Mbps and the propagation delay as 50 ms. We assume the random loss rate is randomly chosen among 1%, 3% and 5%, with changing frequency as 10 Hz. The corresponding results are shown in Table I. It can be observed from Table I that the throughput of DRL-3R significantly outperforms other baselines. Specifically, DRL-3R achieves 63%, 38%, 41%, 28% and 4% improvement on throughput compared with TCP Reno, TCP Westwood, TCP Cubic, TCP BBR and DRL-CC respectively, and reduces about 20-40 ms on RTT. Also note that DRL-3R always achieves a better fairness compared with baselines. In other words, DRL-3R is more robust to the change of random loss rate compared with baselines, improving the throughput while reducing the RTT. It can also maintain a better fairness compared with baselines.

6) Scenario 6: In this scenario, we present the performance of DRL-3R in a network with dynamic CN propagation delay. We set the random loss rate as 1% and the bottleneck bandwidth as 116.8 Mbps. We assume the CN propagation delay is randomly chosen in integers in [50 ms, 60 ms], with changing frequency as 10 Hz. The corresponding results in Table I leads to similar conclusion as to those in Scenario 3. Specifically, DRL-3R achieves an improvement on throughput compared with baselines. Specifically, DRL-3R achieves 64%, 9%, 37%, 14% and 42% improvement on throughput compared with TCP Reno, TCP Westwood, TCP Cubic, TCP BBR and DRL-CC respectively. DRL-3R also achieves a similar RTT

TABLE I
PERFORMANCE COMPARISON OF DRL-3R AND BASELINES IN SCENARIO 4, 5, 6, FTP AND COMPETITION[1]

		Reno	Westwood	Cubic	BBR	DRL-CC	DRL-3R(Avg)	DRL-3R(Best)
Scenario 4 (dynamic bandwidth)	throughput	1.09	1.59	1.29	1.56	1.18	1.66	1.83
	RTT	0.159	0.179	0.161	0.174	0.133	0.168	0.152
	fairness	0.624	0.593	0.604	0.641	0.722	0.689	0.738
Scenario 5 (dynamic loss)	throughput	0.37	0.44	0.43	0.47	0.58	0.61	0.72
	RTT	0.183	0.201	0.184	0.182	0.173	0.160	0.159
	fairness	0.378	0.343	0.379	0.426	0.496	0.577	0.608
Scenario 6 (dynamic delay)	throughput	1.01	1.53	1.21	1.45	1.17	1.66	1.86
	RTT	0.165	0.185	0.169	0.178	0.140	0.176	0.173
	fairness	0.618	0.589	0.600	0.634	0.717	0.684	0.719
Scenario FTP	throughput	1.06	1.56	1.24	1.26	1.20	1.59	1.76
	RTT	0.150	0.166	0.153	0.142	0.131	0.149	0.148
	fairness	0.619	0.603	0.607	0.663	0.721	0.704	0.711
Scenario Competition	throughput	1.10	1.64	1.26	1.63	1.17	1.64	1.82
	RTT	0.156	0.182	0.163	0.176	0.131	0.150	0.143
	fairness	0.619	0.585	0.577	0.632	0.743	0.728	0.773

[1] Throughput, RTT and fairness are all averaged. The unit of throughput is Mbps, and that of RTT is second.

compared with TCP BBR and a high fairness among all TCP flows. Although DRL-CC outperforms DRL-3R on fairness, its low throughput makes its higher fairness index meaningless. In conclusion, in an environment with dynamic CN propagation delay, DRL-3R achieves a higher throughput with similar RTT and a better fairness compared with baselines.

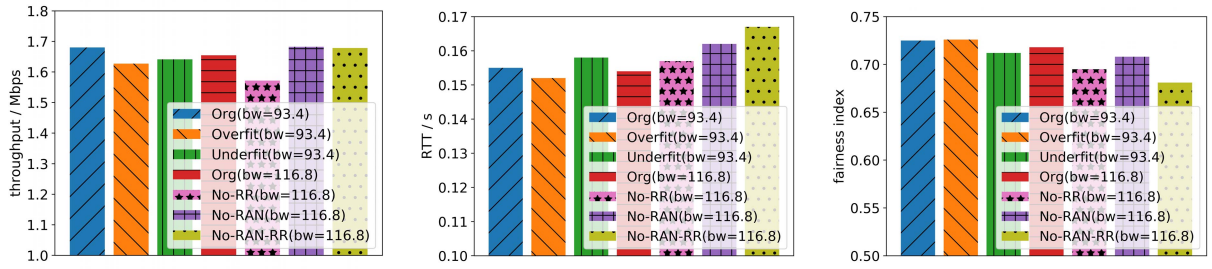
E. Performance Sensitivity Analyses

1) Friendliness: Friendliness is an important feature of congestion control method and we use "friendliness" to judge whether a new method is compatible with previous ones, leaving enough resources for previous methods. To discuss the friendliness of DRL-3R, we implement both DRL-3R and TCP Reno in a scenario similar to Scenario I. The downlink bottleneck bandwidth is 116.8 Mbps. 15 users implement DRL-3R and other 15 flows implement TCP Reno. Experiment results show that DRL-3R achieves an average throughput of 1.71 Mbps, and 1.10 Mbps for TCP Reno, similar to the average throughput when all TCP flows are controlled by TCP Reno, which implies that TCP Reno is allocated reasonable resources and DRL-3R doesn't occupy too much resources of TCP Reno. Finally, we can conclude that DRL-3R is quite friendly to previous methods.

2) Application in FTP Service Model: In our previous experiments, we set the server working in a "bulky" manner. To show the performance of DRL-3R in a more realistic environment, we construct Scenario FTP which is similar to Scenario I, and apply FTP service model to justify the effectiveness of DRL-3R under FTP. We assume that the server has infinite files to transmit. The size of each file follows geometry distribution with $p = 0.05$, and limited in $1 - 100$ segments. The reading time of user follows Pareto distribution (i.e., $p(x) = \frac{am^a}{x^{a+1}}$) with $a = 1.1, m = 10$, and is limited in $10 - 100$ ms. The downlink bottleneck bandwidth is set as 116.8 Mbps. Experiment results are shown in Table I. It demonstrates that DRL-3R also achieves a higher throughput with low RTT compared with other methods, and significantly improve throughput compared with DRL-CC. It also achieves high fairness.

3) Performance Within Fully-Competition Environment: In our previous experiments, the maximum sending rate of each TCP flow is limited as 23.4 Mbps. To allow all TCP flows competing for bandwidth with no limitation, we construct Scenario Competition, in which the maximum sending rate of each TCP flow is not limited, and the downlink outbound bandwidth of the server on each path is very high, allowing all TCP flows sending data as fast as they wish. Experiment results are shown in Table I. We can find that DRL-3R can still achieve superior performance compared with other methods, achieving higher throughput and fairness while maintaining lower RTT, demonstrating that DRL-3R can still work in fully-competition environment.

4) Impact of RAN Information Prediction and Reward Redistribution: In this paper, we apply RAN information prediction network, whose accuracy need to be discussed. Firstly, it should be clarified that the proposed DRL-3R does not require an extremely high accuracy on RAN information prediction, since the purpose of RAN information is to inform the sender of the tendency of RAN to become a bottleneck. In our experiment, the MSE of PRB available ratio and user-received signal strength prediction network are around 0.006 and 7×10^{-7}, which is not extremely high accuracy, but fairly sufficient. To further test the effect of accuracy, we apply DRL-3R with under-fitting and over-fitting pre-trained RAN information prediction network in Scenario I with downlink bottleneck bandwidth as 93.4 Mbps. Experiment results are depicted in Fig. 7 and it can be seen that the fitting status barely exert influences on the performance. Meanwhile, to demonstrate the necessity and effectiveness of reward redistribution, we compare the performance of DRL-3R with and without reward redistribution in Scenario I with downlink bottleneck bandwidth as 116.8 Mbps. Similar comparison is also conducted for DRL-3R, and DRL-3R without RAN information only and DRL-3R without RAN information and reward redistribution to show the effect of RAN information.



(a) Average per-flow throughput. (b) Average RTT. (c) Average fairness index.

Fig. 7. Performance comparison of DRL-3R with different components. "bw" indicates the downlink bottleneck bandwidth the experiment running in. "Org" indicates the original DRL-3R. "Overfit" and "Underfit" indicate DRL-3R with over-fitting and under-fitting pre-trained RAN information prediction network, and same reward redistribution network. Note that the PRB information prediction network used in "Overfit" is de-facto not over-fitting as it doesn't show the tendency of over-fitting until the end of training. "No-RR" indicates DRL-3R without reward redistribution. "No-RAN" indicates DRL-3R without RAN information. "No-RAN-RR" indicates DRL-3R without reward redistribution and RAN information.

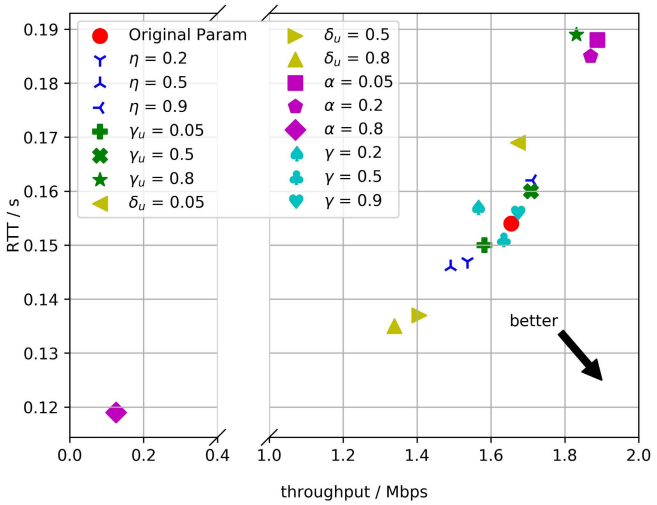


Fig. 8. A comparison of average throughput and average RTT among DRL-3R with different parameter settings. "Original Param" indicates the original parameter setting of DRL-3R, in which $\gamma = 0.99$, $\eta = 0.8$, $\gamma_u = 0.2$, $\delta_u = 0.1$, and $\alpha = 0.5$. For any other parameter setting, these parameters remain identical except the specific parameter pointed out in its legend.

It can be observed from Fig. 7 that after including RAN information, the agent effectively avoids the RAN-induced congestion by slightly reducing the throughput, but significantly reduces RTT. On the other hand, the introduction of reward redistribution can increase about 5% throughput without worsening the RTT. In other words, we can safely come to a conclusion that the incorporation of RAN information and reward redistribution serves to improve network performances. Besides, it can also be observed that although DRL-3R would suffer from utility degradation when RAN information is not available, it can still work well compared with other baselines and well adapt to the environment, proving that DRL-3R has good generality.

5) Effect of Parameters: In this paper, we apply several parameters on DRL-3R. The most important parameters of DRL-3R include γ, η, γ_u, δ_u and α, which directly control the performance of DRL-3R. We test DRL-3R with several different parameter settings. The scenario we test DRL-3R with different parameters in is similar to Scenario I with downlink bottleneck bandwidth as 116.8 Mbps. The experiment result is shown in Fig 8. For discount factor γ defined in Equ. (1), experiment results have demonstrated that it doesn't have significant effect on the performance of DRL-3R. For decay coefficient η defined in Equ. (2), it is positive correlated with throughput and negative with RTT. This is because the changing of η equals changing the importance of throughput in utility function. For fairness parameter α defined in Equ. (4), we set that $\alpha_t = \alpha_r = \alpha$, and its increase leads to a higher throughput with a significantly high RTT, while its decrease produces an extremely low throughput with very low RTT. As the order of magnitude of throughput is significantly higher than that of RTT, the decrease of α would increase the long-term throughput processed by α-fairness function and decrease the RTT. In other words, the changing of α affects the equilibrium between throughput and RTT. For weights γ_u and δ_u[5] for user u defined in Equ. (3), experiment results prove the effectiveness of both of them, which control the relative importance of throughput and RTT. For simplicity, the γ_u and δ_u remain same for all users.

VI. Conclusion and Future Works

In this paper, we have proposed DRL-3R, a DRL based congestion control method. In particular, DRL-3R has incorporated RAN information for congestion control, and embedded a specifically-tailored RAN information prediction module. DRL-3R has relaxed the implicit assumption in other DRL-based congestion control methods by introducing a reward redistribution module, which could be widely applied for communication and networking scenarios with commonly delayed feedback for consecutive actions. Experiment results have demonstrated that DRL-3R yields superior performance in terms of throughput and RTT in various environments. DRL-3R also achieves higher fairness compared with baselines. In our future work, we will explore the application of other kinds of RAN information, implement DRL-3R on more realistic platform (e.g. ns-3), compare DRL-3R with more recent methods, investigate the performance of DRL-3R in

[5]During implementation, we use two constants, which remain unchanged in all experiments, to linearly normalize throughput and RTT respectively while calculating the utility function, to make them in similar order of magnitude and adjustment of γ_u and δ_u more convenient. This trick equals multiplying γ_u and δ_u by two constants respectively.

more scenarios (e.g. wired network, or other kinds of wireless network) and upgrade DRL-3R with multi-agent DRL methods to improve its generality and scalability.

APPENDIX A
PROOF OF EQUATION (10)

We define an SDP with delayed reward $\hat{\mathcal{M}}$ represented by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ with state-action sequence $(\mathbf{s}_0, \mathbf{a}_0), \cdots, (\mathbf{s}_T, \mathbf{a}_T)$, starting at time-step 0, ending at time-step T, with $T+1$ time-steps. At time-step $T+1$, a delayed reward $\hat{R}_{T+1}$ is returned.

According to Equ. (5) and Equ. (8), we have:

$$\begin{aligned} E[R_{t+1}|(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_t, \mathbf{a}_t)] &= \hat{Q}^\pi(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}^\pi(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \\ &= \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \\ &= g(\Delta_{0:t}) - g(\Delta_{0:t-1}) \\ &= \sum_{i=0}^{t} h_i - \sum_{i=0}^{t-1} h_i \\ &= h_t \end{aligned} \tag{13}$$

Finally, we have the conclusion that Equ. (10) has been proved.

APPENDIX B
PROOF OF THEOREM 1

We define an SDP with delayed reward $\hat{\mathcal{M}}$ represented by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ with state-action sequence $(\mathbf{s}_0, \mathbf{a}_0), \cdots, (\mathbf{s}_T, \mathbf{a}_T)$, starting at time-step 0, ending at time-step T, with $T+1$ time-steps. We cut the whole state-action sequence into several shorter sequences with constant length L. Each shorter sequence start at time-step t ending at time-step $t+L-1$, receiving delayed reward $\hat{R}_{t+L}$. Assume that $T+1$ can be divided by L with no remainder. The new SDP $\mathcal{M}$ constructed by reward redistribution (described in Section IV-C) from $\hat{\mathcal{M}}$, has the same state space $\mathcal{S}$, action space $\mathcal{A}$ and state-transition probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ but different reward distribution with $\hat{\mathcal{M}}$. The reward is redistributed to each state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ as R_{t+1}.

Definition 1: the return at time-step t is defined as the cumulative reward starting at time-step t. For SDP $\hat{\mathcal{M}}$ and $\mathcal{M}$, the return can be described as $\hat{G}_t = \sum_{i=0}^{\infty} \hat{R}_{t+i+1}$ and $G_t = \sum_{i=0}^{\infty} R_{t+i+1}$ respectively.

Proposition 1: For each sequence $(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)$, where t is the time-step when a shorter sequence starts, we can get the same expected return for any policy π with delayed reward or redistributed reward.

Proof: For $\hat{\mathcal{M}}$, we have

$$\begin{aligned} &E_\pi[\hat{G}_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= E_\pi[\sum_{i=0}^{\infty} \hat{R}_{t+i+1}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= E_\pi[\sum_{n=1}^{\frac{T-t+1}{L}} \hat{R}_{t+nL}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \end{aligned} \tag{14}$$

For $\mathcal{M}$, we have

$$\begin{aligned} &E_\pi[G_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= E_\pi[\sum_{i=0}^{\infty} R_{t+i+1}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= E_\pi[\sum_{i=0}^{T-t} R_{t+i+1}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \end{aligned} \tag{15}$$

According to the reward redistribution described in this paper, for each shorter sequence $(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_{t+L-1}, \mathbf{a}_{t+L-1})$, we have

$$\hat{R}_{t+L} = \sum_{i=1}^{L} R_{t+i} \tag{16}$$

because our reward redistribution ensures that the sum of redistributed rewards of each sequence equals the delayed reward of this sequence. Similarly

$$\hat{R}_{t+nL} = R_{t+(n-1)L+1} + \cdots + R_{t+nL} \tag{17}$$

In the end,

$$\begin{aligned} &E_\pi[\sum_{n=1}^{\frac{T-t+1}{L}} \hat{R}_{t+nL}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &\qquad = E_\pi[\sum_{i=0}^{T-t} R_{t+i+1}|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \end{aligned} \tag{18}$$

Finally, we have:

$$E_\pi[\hat{G}_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] = E_\pi[G_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \tag{19}$$

Proposition 2: $\hat{\mathcal{M}}$ and $\mathcal{M}$ share same state-action value function.

Proof: According to the definition, the state-action value function of $\hat{\mathcal{M}}$ and $\mathcal{M}$ can be written as $\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) = E_\pi[\hat{G}_t|(\mathbf{s} = \mathbf{s}_t, \mathbf{a} = \mathbf{a}_t)]$ and $Q^\pi(\mathbf{s}, \mathbf{a}) = E_\pi[G_t|(\mathbf{s} = \mathbf{s}_t, \mathbf{a} = \mathbf{a}_t)]$.

Clearly:

$$\begin{aligned} &\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \\ &= E_\pi[\hat{G}_t|(\mathbf{s} = \mathbf{s}_t, \mathbf{a} = \mathbf{a}_t)] \\ &= \sum_{(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \cdots, (\mathbf{s}_T, \mathbf{a}_T)} p_\pi[(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \cdots, (\mathbf{s}_T, \mathbf{a}_T)|(\mathbf{s}_t, \mathbf{a}_t)] \cdot \\ &\quad E_\pi[\hat{G}_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= \sum_{(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \cdots, (\mathbf{s}_T, \mathbf{a}_T)} p_\pi[(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \cdots, (\mathbf{s}_T, \mathbf{a}_T)|(\mathbf{s}_t, \mathbf{a}_t)] \cdot \\ &\quad E_\pi[G_t|(\mathbf{s}_t, \mathbf{a}_t), \cdots, (\mathbf{s}_T, \mathbf{a}_T)] \\ &= E_\pi[G_t|(\mathbf{s} = \mathbf{s}_t, \mathbf{a} = \mathbf{a}_t)] \\ &= Q^\pi(\mathbf{s}, \mathbf{a}) \end{aligned} \tag{20}$$

So, $\hat{\mathcal{M}}$ and $\mathcal{M}$ share same state-action value function.

The optimal policy is defined as the policy that is better than or equal to other policies on state-action value function or state value function. The optimal policy for $\mathcal{M}$, i.e., π_*, maximizes $Q^\pi(\mathbf{s}, \mathbf{a})$, and optimal policy for $\hat{\mathcal{M}}$, i.e., $\hat{\pi}_*$, maximizes $\hat{Q}^\pi(\mathbf{s}, \mathbf{a})$. As $\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a})$ which is proved by Prop. 2, we have the conclusion that $\pi_* = \hat{\pi}_*$.

REFERENCES

[1] M. Chen, R. Li, Z. Zhao, and H. Zhang, "RAN information-assisted TCP congestion control via DRL with reward redistribution," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, Montreal, QC, Canada, Jun. 2021, pp. 1–7.

[2] V. Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3-Reno," in *Proc. Internet Eng. Task Force*, Vancouver, BC, Canada, Sep. 1990, pp. 365–374.

[3] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar./Apr. 2018.

[4] L. Zhang, Y. Cui, M. Wang, Z. Yang, Y. Jiang, and Y. Cui, "Machine learning for internet congestion control: Techniques and challenges," *IEEE Internet Comput.*, vol. 23, no. 5, pp. 59–64, Sep. 2019.

[5] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis, "CQIC: Revisiting cross-layer congestion control for cellular networks," in *Proc. 16th Int. Workshop Mobile Comput. Syst. Appl.*, Santa Fe, NM, USA, Feb. 2015, pp. 45–50.

[6] Y. Xie, F. Yi, and K. Jamieson, "PBE-CC: Congestion control via endpoint-centric, physical-layer bandwidth measurements," 2020, *arXiv:2002.03475*.

[7] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2019.

[8] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. 7th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, Rome, Italy, Jul. 2001, pp. 287–297.

[9] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operat. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

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

[11] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path TCP meets deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1325–1336, Jun. 2019.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[13] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, San Juan, Puerto Rico, May 2016.

[15] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, Aug. 2017, pp. 449–458.

[16] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, Jun. 2016, pp. 2939–2947.

[17] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[18] Z. Wang *et al.*, "Sample efficient actor-critic with experience replay," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, Apr. 2017.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, vol. 5, Stockholm, Sweden, Jul. 2018, pp. 2976–2989.

[20] R. Li, C. Wang, Z. Zhao, R. Guo, and H. Zhang, "The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility," *IEEE Commun. Lett.*, vol. 24, no. 9, pp. 2005–2009, Sep. 2020.

[21] K. Xiao, S. Mao, and J. K. Tugnait, "TCP-Drinc: Smart congestion control based on deep reinforcement learning," *IEEE Access*, vol. 7, pp. 11892–11904, 2019.

[22] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 676–685.

[23] L. Cui, Z. Yuan, Z. Ming, and S. Yang, "Improving the congestion control performance for mobile networks in high-speed railway via deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5864–5875, Jun. 2020.

[24] L. Zhang, K. Zhu, J. Pan, H. Shi, Y. Jiang, and Y. Cui, "Reinforcement learning based congestion control in a real environment," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Honolulu, HI, USA, Aug. 2020, pp. 1–9.

[25] B. He *et al.*, "DeepCC: Multi-agent deep reinforcement learning congestion control for multi-path TCP based on self-attention," *IEEE Trans. Netw. Service Manage.*, early access, Jun. 29, 2021, doi: 10.1109/TNSM.2021.3093302.

[26] M. Bachl, T. Zseby, and J. Fabini, "Rax: Deep reinforcement learning for congestion control," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[27] X. Nie *et al.*, "Dynamic TCP initial windows and congestion control schemes through reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1231–1247, Jun. 2019.

[28] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 11, pp. 2621–2633, Nov. 2019.

[29] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 123–134, 2013.

[30] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2019, pp. 8026–8037.

[31] P. Gawłowicz and A. Zubow, "Ns-3 meets OpenAI gym: The playground for machine learning in networking research," in *Proc. 22nd Int. ACM Conf. Modeling, Anal. Simulation Wireless Mobile Syst. (MSWIM)*, Miami Beach, FL, USA, Nov. 2019, pp. 113–120.

[32] V. Sovrasov. (2019). *Flops Counter for Convolutional Networks in PyTorch Framework*. [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch/

[33] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling*. Hoboken, NJ, USA: Wiley, 1991.

Minghao Chen (Graduate Student Member, IEEE) received the B.E. degree from the School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an, China, in 2019. He is currently pursuing the master's degree with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. His research interests currently focus on deep reinforcement learning and computer communication networks.

Rongpeng Li (Member, IEEE) is currently an Associate Professor with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. He was a Research Engineer with the Wireless Communication Laboratory, Huawei Technologies Company, Ltd., Shanghai, China, from August 2015 to September 2016. He was a Visiting Scholar with the Department of Computer Science and Technology, University of Cambridge, Cambridge, U.K., from February 2020 to August 2020. His research interest currently focuses on networked intelligence for communications evolving (NICE). He received the sponsorship "National Post-Doctoral Program for Innovative Talents" in 2016. He serves as an Editor for *China Communications*.

Jon Crowcroft (Fellow, IEEE) received the degree in physics from the Trinity College, University of Cambridge, Cambridge, U.K., in 1979, and the M.Sc. degree in computing and the Ph.D. degree from University College London, London, U.K., in 1981 and 1993, respectively. Since 2001, he has been the Marconi Professor of communications systems with the Computer Laboratory. He has worked in the area of Internet support for multimedia communications for more than 30 years. Three main topics of interest have been scalable multicast routing, practical approaches to traffic management, and the design of deployable end-to-end protocols. His current research interests include opportunistic communications, social networks, and techniques and algorithms to scale infrastructure-free mobile systems. He is a fellow of the Royal Society, the ACM, the British Computer Society, the IET, and the Royal Academy of Engineering.

Jianjun Wu is the Chief Researcher and the Director of the Future Network Laboratory, Huawei Technologies. He is leading the future network architecture design in Huawei.

Zhifeng Zhao (Member, IEEE) received the B.E. degree in computer science, the M.E. degree in communication and information systems, and the Ph.D. degree in communication and information systems from the PLA University of Science and Technology, Nanjing, China, in 1996, 1999, and 2002, respectively. From 2002 to 2004, he acted as a Post-Doctoral Researcher with Zhejiang University, Hangzhou, China, where his researches were focused on multimedia next-generation networks (NGNs) and soft-switch technology for energy efficiency. From 2005 to 2006, he acted as a Senior Researcher with the PLA University of Science and Technology, where he performed research and development on advanced energy-efficient wireless router, *ad-hoc* network simulator, and cognitive mesh networking test-bed. From 2006 to 2019, he was an Associate Professor with the College of Information Science and Electronic Engineering, Zhejiang University. Currently, he is with the Zhejiang Lab, Hangzhou. His research areas include software defined networks (SDNs), wireless network in 6G, computing networks, and collective intelligence. He is the Symposium Co-Chair of ChinaCom 2009 and 2010. He is the Technical Program Committee (TPC) Co-Chair of the 10th IEEE International Symposium on Communication and Information Technology (ISCIT 2010).

Honggang Zhang (Senior Member, IEEE) was an Honorary Visiting Professor with the University of York, York, U.K., and an International Chair Professor of excellence with the Université Européenne de Bretagne and Supélec, France. He is a Full Professor with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. He has coauthored and edited two books: *Cognitive Communications: Distributed Artificial Intelligence (DAI), Regulatory Policy & Economics, Implementation* (John Wiley & Sons) and *Green Communications: Theoretical Fundamentals, Algorithms and Applications* (CRC Press), respectively. His research interests include cognitive radio and networks, green communications, mobile computing, machine learning, artificial intelligence, and the Internet of Intelligence (IoI). He is a co-recipient of the 2021 IEEE Communications Society Outstanding Paper Award and the 2021 IEEE INTERNET OF THINGS JOURNAL (IoT-J) Best Paper Award. He was the leading Guest Editor for the Special Issues on Green Communications of the *IEEE Communications Magazine*. He served as a Series Editor for the *IEEE Communications Magazine* (Green Communications and Computing Networks Series) from 2015 to 2018 and the Chair of the Technical Committee on Cognitive Networks of the IEEE Communications Society from 2011 to 2012. He is the Associate Editor-in-Chief of *China Communications*.