Safety in DRL-Based Congestion Control: A Framework Empowered by Expert Refinement

Jianer Zhou, Zhiyuan Pan, Zhenyu Li, Gareth Tyson, Weichao Li, Xinyi Qiu, Heng Pan, Xinyi Zhang, and Gaogang Xie

***Abstract*—Deep reinforcement learning (DRL) has been used in congestion control algorithms (CCAs) for its ability to adapt to different network environments. However, its effectiveness is often hindered by the limited availability of training data and constrained training scales. While it has been proved that combining rule-based (expert) CCAs as a guide for DRL (namely hybrid CCAs) can address this limitation, we show through experimental measurements that rule-based CCAs potentially restrict action exploration of DRL models and may cause the DRL models to overly rely on them for higher reward gains. To address this gap, this paper proposes Marten, a framework that improves the effectiveness of rule-based CCAs for DRL. Marten's key innovations include an entropy-based dynamic exploration scheme that expands the exploration of DRL, and a reward adjustment scheme to prevent the DRL models' over-reliance on experts in hybrid CCAs. We have implemented Marten in both simulation platform OpenAI Gym and deployment platform QUIC. Experimental results in both emulated and production networks demonstrate Marten can improve throughput by 0.31% and reduce latency by 12.69% on average compared to the state-of-the-art hybrid CCAs. Compared to BBR, Marten achieves a 2.79% increase in throughput and an 11.73% reduction in latency on average.**

***Index Terms*—Congestion control, machine learning, deep reinforcement learning, QUIC.**

I. Introduction

THE mainstream congestion control algorithms (CCAs) are divided into three categories: rule-based algorithms, learning-based algorithms and hybrid algorithms. Rule-based algorithms, such as Cubic [1], BBR [2], C2TCP [3], DCTCP [4], Copa [5], TIMELY [6], pFabric [7] and Sprout [8], are designed for specific network scenarios. These algorithms have been proved to have convergence ability and fairness, predictable behavior and low computational overhead. However, such algorithms can only work in specific scenarios and cannot meet the needs in new network scenarios. Learning-based algorithms, such as Aurora [9], MOCC [10], AUTO [11] and DeepCC [12], rely on deep reinforcement learning (DRL) agents as the control core, as DRL's feedback mechanism is most similar to that of congestion control. It has been shown that, on average, such algorithms have lower delay and their overall performance is better than rule-based algorithms. Meanwhile, these algorithms can be deployed in many network scenarios to achieve generalization. However, due to the exploration-based mechanism of DRL, the model may make mistakes or unsafe actions, resulting in the long tail phenomenon of latency. The long tail effect will cause poor performance. In order to overcome the problems of DRL, hybrid algorithms come into being, such as Eagle [13], Orca [14] and Libra [15]. Through the fusion of rule-based and learning-based framework, the hybrid algorithms use rule-based algorithm (such as BBR or Cubic) as an expert to guide the DRL model, so that the overall framework can inherit the advantages of both kinds of CCAs: stability, fast convergence and fairness (rule-based); adaptability and high performance (learning-based).

The purpose of hybrid algorithms is to mitigate the severe consequences of incorrect actions by DRL models and to enhance convergence speed. However, our performance diagnosis experiments reveal two key limitations in existing hybrid CCAs.

The first limitation is that the exploration capability of a DRL model guided by rule-based CCAs is constrained, leading to a performance ceiling. As a result, the DRL model inevitably converges to the rule-based CCA, preventing it from achieving high performance in new or dynamic network scenarios. The second limitation is the DRL model's potential over-reliance on rule-based CCAs, which may lead to incorrect actions. If the DRL model fails to identify the correct action during exploration, the shielding mechanism is triggered to mitigate the impact of incorrect actions by reverting to rule-based algorithms. In such cases, the reward provided by the

Received 9 August 2023; revised 1 January 2025; accepted 29 May 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor M. Zhang. Date of publication 24 June 2025; date of current version 18 December 2025. This work was supported in part by the Major Key Project of PCL under Grant PCL2025A02 and Grant PCL2024Y02, in part by the PCL-CMCC Foundation for Science and Innovation under Grant 2024ZY1A0010, and in part by the Young Scientists Fund of the National Natural Science Foundation of China under Grant 62202447. *(Jianer Zhou and Zhiyuan Pan are co-first authors.) (Corresponding author: Zhenyu Li.)*

Jianer Zhou, Weichao Li, and Xinyi Qiu are with the Pengcheng Laboratory, Shenzhen 518066, China (e-mail: zhouje1005@gmail.com; liwc@pcl.ac.cn; qiuxy@pcl.ac.cn).

Zhiyuan Pan is with the Southern University of Science and Technology, Shenzhen 518055, China, and also with Tencent, Shenzhen, China (e-mail: 12133090@mail.sustech.edu.cn).

Zhenyu Li is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100045, China (e-mail: zyli@ict.ac.cn).

Gareth Tyson is with The Hong Kong University of Science and Technology, Guangzhou, China (e-mail: gtyson@ust.hk).

Heng Pan, Xinyi Zhang, and Gaogang Xie are with the Computer Network Information Center, Chinese Academy of Sciences, Beijing 100045, China (e-mail: panheng@cnic.cn; xyzhang@cnic.cn; xie@cnic.cn).

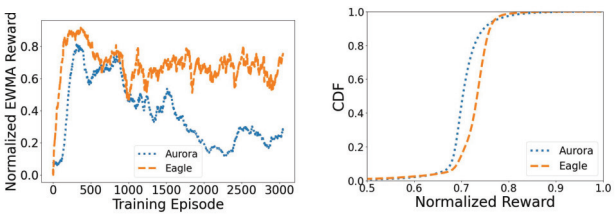
Digital Object Identifier 10.1109/TON.2025.3580436

rule-based algorithm is often higher, leading to an increase in the model's average reward. Due to the DRL model's inherent pursuit of maximizing expected rewards, it may deliberately take incorrect actions to repeatedly activate the shielding mechanism. This behavior is clearly flawed and counterproductive.

These two constraints limit the performance of hybrid algorithm. The root cause of these two limitations is that the existing DRL training process gradually reduces the curiosity of the model about the environment. However, given the variation of the network scenarios, continuous reduction of action space exploration would limit the exploration space of DRL, leading to sub-optimal performance. To address these challenges, we propose Marten, a hybrid CCA framework designed to overcome the constraints of rule-based CCAs imposed to DRL. Marten includes two key innovations to address the identified limitations. First, Marten introduces *entropy* as a novel metric to represent the DRL model's curiosity about its environment. Unlike constant or gradually decreasing exploration strategies, Marten leverages entropy to dynamically increase the model's exploration capacity under specific network conditions, enabling it to quickly identify more effective strategies. Second, when the DRL model's actions trigger guidance from rule-based experts, Marten modifies the reward function for such actions to reduce the model's over-reliance on expert algorithms. This adjustment ensures that the DRL model continues to learn independently and avoid excessive dependence on external guidance. We demonstrate that these two innovations are effective measures to enable the DRL model to learn safely and adaptively.

We have implemented Marten in both simulation platform OpenAI Gym [16] and deployment platform QUIC [17]. QUIC is a multiplexed transport protocol implemented in user space and has become the standard transport layer for HTTP/3 [18]. Through extensive experiments, we demonstrate that Marten effectively maintains the safety of the DRL model while expanding action exploration and achieving higher performance. Our key contributions are:

- Through experimental measurements, we reveal that in hybrid CCAs, the expert (rule-based) introduces two constraints to the DRL training model: limited exploration (effectiveness) and excessive reliance on experts (safety). These constraints significantly limit the performance of hybrid CCAs.
- We propose Marten, a framework that dynamically adjusts the exploration of the DRL model by using entropy to assess the degree of exploration, while still benefiting from the guidance of rule-based CCAs. Our experimental results demonstrate that adjusting the exploration degree enhances the DRL model's ability to converge to better solutions.
- We have implemented Marten on both the simulation platform OpenAI Gym and the deployment platform QUIC. Deployment experiments conducted over both emulated and production networks show that Marten effectively reduces latency by 12.69% while improving throughput by 0.31% on average compared with the state-of-the-art hybrid CCAs (Eagle and Libra). Compared with BBR, Marten improves throughput by 2.79% and reduce latency by 11.73% on average.



(a) EWMA (b) CDF

Fig. 1. Comparison of the EWMA and CDF reward value for Aurora and Eagle.

The rest of the paper is structured as follows. Section II introduces our motivation and challenges. Section III shows the overview of the system. Section IV presents the design detail of Marten. Section VI explains the implementation of Marten and our extensive experiments. Related work is covered in Section VIII and Section IX concludes the paper.

II. Motivation and Challenges

Experiments for existing hybrid CCAs reveal two constraints of existing hybrid CCAs: DRL exploration limitation and excessive reliance on experts. In this section, we will explain the constraints in detail and discuss the challenges for solving them.

A. Motivation

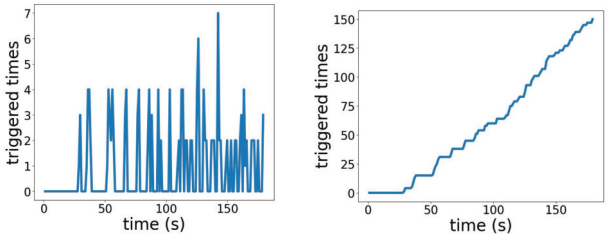
We implement Eagle [13], which uses BBR as an expert to guide DRL and Aurora [9] which uses pure learning-based method. We test them in the OpenAI Gym, which can simulate a series of network scenarios through four parameters: link capacity (1Mbps-80Mbps), minimum RTT (20-500ms), buffer size (0.5-5BDP), and stochastic loss rate (0-5%). For each network scenario, we randomly select network parameters from the above range, and keep the parameters unchanged in this scenario. More details of the settings of the experiments will be introduced in Section VI-A.

In order to intuitively compare the convergence capabilities of different algorithms, we train the model from clean-state. As shown in the Figure 1a, we confirm that hybrid CCAs have the advantages of fast convergence and relative stability compared with the learning-based CCAs. After 100 episodes, Eagle [13], a hybrid CCA, reaches the EWMA (exponentially weighted moving average) reward that Aurora requires 300 episodes to reach. The results imply that Eagle can find better policy faster and keep the high reward value. Note that, we use EWMA as a measure of reward value as it can smooth the measured value to better show trends. After 1,100 episodes, the EWMA reward of Aurora continues to decrease, indicating that the pure DRL algorithm is unstable in some scenarios. We conclude two main problems of the existing hybrid CCA through experiments: DRL exploration limitation and excessive reliance on experts.

DRL exploration is limited by rule-based CCAs. We test 8,000 different network scenarios. Each network scenario

TABLE I
QUANTILE COMPARISON OF NORMALIZED REWARD

reward	aver.	5%	10%	20%	90%	95%
Aurora	0.71	0.65	0.69	0.72	0.75	**0.77**
Eagle	**0.73**	**0.67**	**0.70**	**0.75**	**0.76**	0.76



(a) Triggered times per second (b) Cumulative triggered statistics

Fig. 2. Number of the rule-based (expert) algorithm triggered.

has different bottleneck link bandwidth, RTT, packet loss rate, and buffer queue length in our simulated environment. Figure 1b shows the CDF (cumulative distribution function) of the normalized reward values of all scenarios. We are surprised to find that while Eagle's average reward is higher than Aurora, its tail performance is not satisfactory. Specific values can be seen in Table I. This phenomenon can be attributed to the fact that DRL model actions are assimilated by the rules and algorithms guided, resulting in inadequate exploration of action space. Existing DRL algorithms reduce the exploration effort over time, while the expert may restrict DRL to make some actions.

Rule-based CCAs induce DRL's excessive reliance. Reinforcement learning algorithm is based on exploration algorithm. When making action decisions based on strategies, there would inevitably be some incorrect behaviors. The consequences of these incorrect behaviors are disastrous for CC as the wrong actions have continuous effect on subsequent actions. The hybrid algorithm aims to use rule-based algorithms (experts) to mitigate the impact of these erroneous operations.

Through experiments, we found that at the early stage of the training model, the reward for the model to take action according to its own strategy is not as effective as the benefit triggered by the expert algorithm. So DRL would frequently make unsafe actions that can be controlled by experts to obtain a higher average reward value. This is evidenced in Figure 2, which shows the number of times that the expert is called in a wired network where the bottleneck link and minimum RTT remain unchanged. The increasing call frequency of the expert algorithm suggests that the DRL model experiences a high number of erroneous behaviors and experiencing incorrect convergence. Note that, there are much more triggers of the expert in wireless networks due to network fluctuations. This highlights our key motivation: expert algorithms contribute to the DRL model's excessive reliance on them.

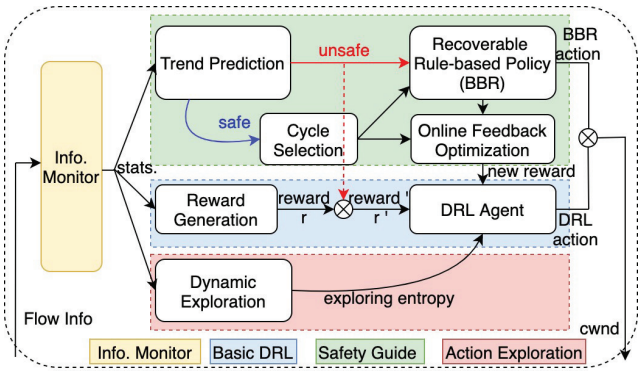


Fig. 3. High-level block diagram of Marten.

B. Challenges

Enlarge exploration while keeping the guidance. To prevent limiting the exploration ability of DRL models, we need to decide how to increase or decrease the ability to explore the action space. In addition, the degree of exploration is difficult to deal with, because it needs to balance the safety and effectiveness of the algorithm. Small exploration limits the overall performance of the model, resulting in the convergence of the model to a sub-optimal solution. Large exploration leads to behavior inconsistent with the current policy, and there is a risk of error.

Correctly identify the wrong actions of DRL. It is challenging to identify whether the change of network statistics is caused by the change of network itself or the wrong action of DRL. In some network scenarios, the network itself is constantly fluctuating. We hope to identify the wrong actions of DRL and introduce experts to guide DRL at the appropriate time.

Avoid DRL's dependence on expert. How to use expert to guide the convergence is also a problem worth discussing. On one hand, we hope that the expert can properly interfere with the online training of DRL, so as to effectively guide DRL to converge to a stable state quickly. On the other hand, we also hope that expert does not guide the DRL too frequently to prevent DRL from relying excessively on experts and losing its own judgment.

III. OVERVIEW OF MARTEN

To overcome the two problems described in Section II (namely DRL exploration limitation and excessive reliance on expert), we design Marten, a framework that improves the effectiveness of rule-based CCAs for DRL. Figure 3 shows Marten's overall architecture. Marten introduces Action Exploration and Safety Guide components respectively. The other components Information Monitor and Basic DRL constitute the basic process of DRL algorithm.

The Information Monitor collects all flow information within a Monitor Interval (MI), processes and analyzes the information, and obtains statistics that reflect the current network condition. The Safety Guide component judges whether the learning-based action leads to unsafe actions and then uses rule-based policy to guide the Basic DRL component. The

Algorithm 1 Marten

```
Input : The flows' information.
Output: The cwnd for each flow.
for cwnd decision do
    //Cycle Selection for algorithm.
    if Cycle Selection to expert then
        Using the cwnd from BBR;
    else
        //Cycle Selection to DRL.
        //Safety guide. (Section IV.C)
        if (Enter unsafe state) then
            Trigger shielding mechanism;
            CC_BBR is called for control;
            Adjusting the experience placed in the
             experience pool;
        //Dynamic exploration. (Section IV.D)
        if (CC_BBR is called) then
            Enter unsafe increase state;
            Increase the exploration entropy of DRL
             model;
        if (Expert is not called for a long time) then
            Enter safe increase state;
            Increase the exploration entropy of DRL
             model;
        else
            Enter natural attenuation state;
            Reduce exploration entropy;
        // Adjust the CC action.
        Change DRL's action according to entropy;
        Calculate cwnd for each flow based on each
         modified actions;
return cwnd
```

Action Exploration component calculates the value of entropy (to reflect the curiosity of DRL model to the environment) in real time, and guides DRL to explore more effective actions based on entropy. The Basic DRL component obtains both reward from the Reward Generation block and entropy from Action Exploration component, and makes corresponding decisions based on modified DRL algorithm.

IV. Design in Detail

In this section, we describe Marten's four components in detail. Subsection IV-A and IV-B detail how Marten collects TCP information and the basic DRL algorithom that Marten uses. Subsection IV-C and IV-D introduce Marten's key components: Safety Guide and Action Exploration. We use Algorithm 1 to briefly show the main process of Safety Guide and Action Exploration and how Marten calculates the *cwnd* based on them.

A. Information Monitor

The main task of Information Monitor component is to collect the flow information, summarize the data within one MI and send them to other blocks. Information Monitor component provides real-time information for the whole algorithm platform. The information can be divided into two kinds: measured statistics (shown in Table II) collected directly from CC platform (Linux or QUIC) and the normalized statistics (shown in Table III) calculated by the measured statistics. We distinguish each flow by five tuples, namely server IP address, server port number, user IP address, user port number and transport layer protocol.

TABLE II
Measured Statistics

Name	Brief Description
thr	The average throughput in this MI
d	The average latency in this MI
$loss$	The average loss rate in this MI
thr_{max}	The maximum value of throughput in the past
d_{min}	The minimum value of latency in the past
num	The number of ACKs in this MI
$cwnd$	The cwnd value in last MI

TABLE III
Normalized Statistics

Name	Brief Description
thr_c	thr/thr_{max}
d_c	d_{min}/d
$\bigtriangledown d$	Derivative of latency with respect to time
$loss_c$	$5 * loss/thr_{max}$
$cwnd_c$	$num/cwnd$
d_{metric}	$1.25 * d_{min}/d\ (if\ d > 1.25 * d_{min})$ $or\ 1\ (if\ d \leq 1.25 * d_{min})$

The basis for solving a reinforcement learning problem is to construct a Markov decision process [19]. Nevertheless, different networks do not necessarily strictly possess Markov characteristics. Therefore, we use the regression structure. Network congestion control is essentially a Partially Observable Markov Decision Process (POMDP) [20]. DRL agent cannot directly access the information of network status, but can only infer from the collected network data. We aggregate the statistics collected in the past and current, and enhance the inference ability of the agent through the correlation between the data. The final feature of the output of Information Monitor to other components is: $s_t = (o_t, o_{t-1}, \ldots, o_{t-m})$, where o_t represents the observed states value at time *t*, and o_{t-i} is the *i*-th observed block nearest to now. *m* determines how many historical information are collected and by default we set *m* to 10.

B. Basic DRL

Now, we describe the core component of DRL agent. The DRL agent observes a series of network states $s \in S$ provided by the Information Monitor, makes corresponding action $a \in A$ according to the strategy $\pi : S \rightarrow A$, and obtains reward values $R_t = \sum_{i=t}^{T} \gamma_{i-t} r(s_i, a_i)$ from the environment, where

γ is discount factor $\in$ (0,1]. The main task of DRL is to find the optimal strategy π_ϕ, with parameters ϕ to maximize the expected reward return value $J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_0]$ [19][21].

Action. The ultimate output of CCAs is congestion window (cwnd) and pacing rate. CCAs use both cwnd and pacing rate to control the packet sending rate. The pacing rate determines when (how quickly) to send out the packets while cwnd determines how many packets can be in flight. Our solution does not adjust the pacing rate, but focuses on the cwnd to control the amount of packets in flight. We expect the action space to be continuous, so that the agent has smooth and fine-grained control. Libra [15] analyzes the three main action spaces, and concludes that exponential growth makes the action to be unlimited and respond quickly. Inspired by Libra we calculate the cwnd by:

$$x_{t+1} = x_t * 2^{a_t}, \quad a \in (-2, 2) \tag{1}$$

x_t is the cwnd at time t, and a_t is the action DRL agent made for next time. Marten achieves the continuity by derived the action at $t+1$ from the action at time t.

Reward. We train Marten through a linear reward function which achieves high throughput, low latency and less packet loss at the same time. A linear function is simple yet effective.

$$\omega_1 * \frac{throughput_t}{throughput_{max}} - \omega_2 * \frac{latency_t}{latency_{min}} - \omega_3 * loss \tag{2}$$

The ω_1, ω_2 and ω_3 are design parameters which can define throughput, latency and packet loss's influence respectively and their values would be defined in Section V.

Learning Algorithm. DRL algorithm in Marten is based on twin delayed deep deterministic policy gradient algorithm (TD3) [22]. TD3 introduces two critical networks to emulate the double-Q network to solve the over-estimation of the action's Q value. The update of actor is delayed, which makes the training of actor more stable. Noise is added to the target actor to increase the stability of the algorithm. The predecessor of TD3 is deep deterministic policy gradient (DDPG) [23], which uses actor-critic framework to concurrently learn policy (actors) and action-value functions (critics). Both of them are implemented by deep neural networks. The action space of the DDPG algorithm is continuous, the same as TD3. In congestion control, continuous action space is more natural than discrete action space, which can better exert the fine control of reinforcement learning model.

To solve the continuous action problem under the framework of actor-critics, actor can use the gradient of expected reward to update its strategies Readers are referred to [24] for details of deterministic policy gradient algorithms..

$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi}[\nabla_a Q^\pi(s,a)|_{a=H(\pi(s))} \nabla_\phi \pi_\phi(s)] \tag{3}$$

where H is additional entropy function that changes actor' actions based on entropy, the degree of exploration which will be described in Eq. 17 in Section IV-D.

The critic estimates the Q value according to the pair of state and action, that is:

$$Q^\pi(s,a) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim H(\pi)}[R_t|s,a]$$
$$= r + \gamma \mathbb{E}_{s',a'}[Q^\pi(s',a')], \quad a' \sim H(\pi_\phi(s_t)) \tag{4}$$

where a' and s' are subsequent state-action pair.

The mechanism of maximizing the noisy value estimation leads the critic to overestimate Q value, which causes inaccurate estimates. We adopt clipped double-Q learning from TD3 algorithm to solve this problem. Two independent (but common experience pool) critics are used to learn the Q of action-value function, and the smaller value is taken as the final update. The target function can be written as:

$$y = r + \min_{i=1,2} Q_{\theta_i'}(s', \pi_{\phi'}(s') + \epsilon) \tag{5}$$

In order to enhance the stability of the model, the network is updated from the temporal difference of another frozen target network $Q_{\theta'}(s,a)$ to ensure that the objective function remains stable during several updates.

C. Safety Guide

Marten uses the safe learning mechanism to ensure the safety of action made by DRL.

1) Definition of Safe Learning: Assuming that the pure DRL decision is Π_{drl}. Safe learning guarantees the safety of the overall policy by introducing a protective policy Π_{shield}, and minimizes the modification of Π_{drl}. Π_{shield} is composed of a DRL-based policy Π_{drl} and a rule-based tradition policy Π_{backup}. As shown in Eq. 6, Π_{shield} will choose Π_{drl} when the next state $f(s,a)$ observing s and doing a following policy π is in invariant states. If next state $f(s,a)$ is unsafe, then backup policy would be used to recover.

$$\Pi_{shield} = \begin{cases} \Pi_{DRL}, & iff(s,a|a = \pi(s)_{DRL}) \in S_{inv} \\ \Pi_{backup}, & otherwise \end{cases} \tag{6}$$

There are two key requirements when using Π_{shield}: 1) Use Π_{DRL} as much as possible. 2) Ensure that the Π_{shield} is safe all the time.

2) Classification of States: There are three kinds of states in safe learning framework, which are invariant state, safe state, unsafe state.

Definition 1: Invariant state. Invariant state means that current state observed from the environment can still stay in a safe state after taking *any* action.

$$S_{inv} = \{s \in f(s,a) | \exists a \in A, s.t.(s,a) \in Z_{eq} \subseteq S_{safe} * A\} \tag{7}$$

Definition 2: Safe state. Safe state means that current state observed from the environment can stay in a safe state after algorithm takes *safe* actions. If the action taken by the algorithm is unsafe, unsafe states will occur.

Given $S_{safe} \in S$, $S_0 \subseteq S_{inv}$, $\forall s \subseteq S_0$, if

$$P_{a \sim P_A}(f(s,a|a = \pi(s)_{drl}) \subseteq S_{safe}) \geq 1 - \varepsilon \tag{8}$$

then we call Π is ε-safe.

Definition 3: Unsafe state. Unsafe state refers to that if the current observed state (feedback from the network) enters the range of unsafe features (such as the sharp decline of RTT or bandwidth), it will lead to a significant decline in overall

performance. Unsafe state is unavoidable even in optimal DRL model, so we need to use safe guide to correct them into safe state.

3) Safe Guide Component: We now describe the Safe Guide Component in Figure 3.

Trend Prediction. In the view of the network unpredictability, it is essential to identify the disastrous phenomena caused by DRL error actions. The function of the trend prediction module is to judge whether there is a serious network error by combining the past and current network status. Considering the jitter of the network itself, the dynamic threshold method is adopted in the scheme design.

Specifically, the threshold is calculated as:

$$Thr_t = \frac{1}{k}\sum_{i=t-k}^{t-1} R\bar{T}T_k \tag{9}$$

where k is the length of history information. A large k means we can use more history information but at the cost of more computation resource. By default, we set k as 20.

Marten uses a sign D to mark unsafe state, which is:

$$D = \begin{cases} true, & \rho * R\bar{T}T_t > Thr_t \\ false, & otherwise \end{cases} \tag{10}$$

If D is true, it is considered that the current state is unsafe. The experts will be called for control in the next MI, and the reward value of the current MI will be punished. It is worth noting that we use the change of *RTT* as the signal to decide whether the state is unsafe, because it is more sensitive to the queue inflation in the bottleneck links than the available bandwidth or packet loss rate. As such, it captures the real-time network status more accurately than other metrics. We will discuss the unsafe state and the value of ρ in Section VI-E. By default, we set ρ as 1.5.

Recoverable Rule-based Policy. Π_{shield} is composed of Π_{DRL} and Π_{backup}. If the agent can successfully find an action from Π_{backup} after making a series of unsafe actions, enabling the system to re-enter a safe state after several actions, then we call this Π_{DRL} repairable. In detail, Π_{DRL} is repairable and can find safe behavior at time t if and only if for all $i < t$, $s_i \in S_{unsafe}$ and at time t, $s_t \in S_{safe}$.

Marten introduces the rule-based algorithm BBR as an expert, because BBR algorithm is stable and widely used. Unlike loss-based algorithms, BBR controls cwnd through bandwidth delay product (BDP) [25]. When the parameter D in Eq. 10 is true, BBR will repair the congestion window as a backup policy. Marten leaves a certain amount of detection opportunities for BBR to detect available BDP, so BBR can adjust the transmission rate to a BDP in a short time.

Cycle Selection. In the process of expert guidance, it is necessary to ensure the principle of minimum intervention, and only intervene when the agent is in a unsafe state. Appropriate intervention is also needed to ensure the safety of the overall system and ensure that the model will not make frequent errors due to the exploration of characteristics at the initial stage of training. When designing cycle selection, we consider these requirements.

By default, we set the total length of cycle selection to five MIs. At the beginning of training, the DRL model controls the first, third and fifth MIs, and the BBR controls the remaining two MIs. In the test process, since the model has learned a lot of knowledge, we change the control frequency of BBR to half of the original, that is, only one of the five MIs. It is worth mentioning that when the trend prediction block finds that there is a risk of error in the detection cycle of reinforcement learning control, BBR will temporarily replace the DRL model to control the next MI.

Online Feedback Optimization. We use Replay memory M to store the DRL experience such as state, action and reward, which can be defined as a multivariate group $e_t = (s_t, a_t, r_t, s_{t+1})$. Putting the experience of experts directly into M seems to be a solution to make the model converge faster. But in fact, it further aggravates the restrictions of expert behavior on the model, as it makes the model more likely to converge to a sub-optimal solution close to expert behavior. To address this problem, Marten changes the process of storing the experience into M as follows.

Once Marten triggers the MI's shielding mechanism, it indicates that such action is unsafe or undesirable, so we should further reduce the reward function of MI_{t-1} as Eq. 11.

$$r'_{t-1} = \begin{cases} \sigma_1 * r_{t-1}, & r_{t-1} \geq 0 \\ \sigma_2 * r_{t-1}, & r_{t-1} < 0 \end{cases} \tag{11}$$

At the same time, we take the behavior of the expert at MI_{t-1} as a guide and store it into DḂy shielding mechanism, the expert controls MI_tİn order to prevent the model from relying on experts, the reward function of MI_t also needs be appropriately reduced as Eq. 12:

$$r'_t = \begin{cases} \theta_1 * r_t, & r_t \geq 0 \\ \theta_2 * r_t, & r_t < 0 \end{cases} \tag{12}$$

σ_1, σ_2, θ_1, θ_2 are design parameters and we will discuss the choice of value in Section VI-E. By default, we set σ_1, σ_2, θ_1 and θ_2 as 0.5, 1.5, 0.75, 1.25 respectively.

4) Analysis: Next we analyze two properties of the Safe Guide Component. First, The trajectory of the Π_{shield} is safe. This can be easily derived as follows. Suppose at time t, the trend prediction block takes D as false, meaning that Π_{DRL} is safe; then Π_{shield} = Π_{DRL} and s_t is safe. Otherwise, if D is true, then $\Pi_{shield} = \Pi_{backup}$; then with the safe guide, Marten can guaranteed that the next state s_{t+1} is safe.

Second, using $a_t^{\Pi_{shield}}$ can improve efficiency of model training and convergence. Let's take Q_1^π as the value function of π_1 with $e = (s, a_t^{\Pi_{shield}}, r', s')$, Q_2^π as the value function of π_2 with $e = (s, a, r, s')$. Let's further assume $Q^{pi}(s) = -\infty, for\ \forall s \in U_{t\leq k}D_t$. Then, we have:

$$\begin{aligned} Q_1^\pi(s) &= E_{a\sim\Pi_{shield}}[R(s, a_t^{\Pi_{shield}}) + \gamma E_{s'}Q_1^\pi(s')] \\ &= E_{a\sim\Pi_{shield}}[r'_t + \gamma E_{s'\notin U_t\leq k}P'(s,a,s')Q_1^\pi(s') \\ &\quad + \gamma E_{s'\in U_t\leq k}P'(s,a,s')r'_{t+1}] \\ Q_2^\pi(s) &= E_{a\sim\Pi}[R(s,a) + \gamma E_{s'}Q_2^\pi(s')] \\ &= E_{a\sim\Pi}[R(s,a) + \gamma E_{s'\notin U_t\leq k}P'(s,a,s')Q_2^\pi(s') \\ &\quad + \gamma E_{s'\in U_t\leq k}P'(s,a,s')(-\infty)] \end{aligned} \tag{13}$$

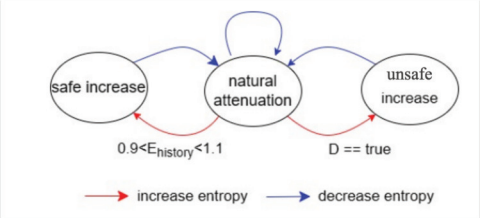


Fig. 4. Flow chart for the change of entropy.

$$= -\infty \tag{14}$$

A finite $Q_1^\pi(s)$ implies that the model with shielding can learn how to make correct behavior. This implies that taking Π_{shield} action can improve the convergence of the model. On the other hand, as $Q_2^\pi(s)$ is negative infinity, the model without shielding can only learn that the current action is erroneous; this is less instructive than $Q_1^\pi(s)$ and will result in longer time for convergence.

D. Dynamic Exploration

Marten uses the Dynamic Exploration Component to dynamically adjust the exploration ability of DRL model for the environment. We propose to use the concept of entropy to measure the curiosity of DRL model to the current environment. In information theory, entropy is a concept to evaluate the state of disorder and uncertainty. A larger entropy means the state is more disordered and contains more information. In our mechanism, we follow the same principle: when we want to have more exploration actions to expand the system, we will increase the entropy; otherwise we will decrease the entropy. It is worth noting that a large entropy may impose a large disturbance on the basis of the actors' actions, while with a small entropy, the dynamic exploration component interferes with the actors' actions less.

Figure 4 shows the entire process of entropy change. We have designed a three-state state machine, with three states: *unsafe increase, safe increase, and natural attenuation.*

The *unsafe increase state* represents that DRL model triggers the shielding mechanism of expert algorithms, thus believing that unsafe behavior has occurred in the current network. This behavior may be caused by incorrect actions from current DRL model, or it may be due to switching or fluctuations. In such a unsafe situation, Marten chooses to increase entropy to improve the strategy updating ability and exploration ability for action space. The *safe increase state* represents that the model has not triggered intervention from expert algorithms for a long period of time, indicating that the DRL model has basically converged, but there is still a possibility of the model converging to a sub-optimal solution. In order to better enable the model to find the optimal solution, Marten chooses to increase entropy appropriately within a limited range to fully explore the action space, and attempted to break the limitations brought by expert algorithm guidance. The *natural attenuation state* is the normal state of the state machine, and it is necessary to appropriately reduce the entropy value to ensure the convergence of the model strategy. After each action taken by the model, regardless of whether it enters a state of increased danger or increased safety, it will eventually return to a natural attenuation state. In a natural attenuation state, the entropy value will decrease according to the pre-designed strategy, making the model tend to converge.

In the specific implementation process of state machines, there are still two major challenges: how to change entropy and how to modify DRL's action based on entropy.

The first challenge is when to increase entropy to improve the ability of the model to find better actions and when to reduce it to make the model converge gradually.

Marten increases entropy in two states, namely unsafe increase state and safe increase state. In unsafe increase state, we used *D* (shown in EQ 8) as a symbol of whether the DRL model action is safe. When *D* is true, the state machine enters unsafe increase state because DRL model is in the wrong convergence process. Under the safety learning framework guided by experts, Marten punishes unsafe actions and increases the entropy, so that the model can find more appropriate actions to adapt to current scenarios. In safe increase state, Marten increases entropy in moderation to find a better policy within a relatively safe situation, trying to break through the limitations brought by expert guidance. Marten uses $E_{history}$, a FIFO (First In First Out) array with length of 20, to store E_t, where E_t represents the ratio between RTT_t and average value of historical *RTT* value. It can be calculated by Eq. 15:

$$E_t = RTT_t / Thr_t \tag{15}$$

The safe increase will be triggered when all 20 values in $E_{history}$ are within the range of 0.9 to 1.1.

Marten reduces the entropy in natural attenuation state. In order to make sure DRL model tends to converge, a certain entropy value will be reduced according to a specific proportion. We use *e* to refer to the numerical value of entropy. e_0 is fixed as 1. e_t represents the entropy value of the MI_t. The recursive calculation method of entropy value is shown as

$$e_t = \begin{cases} (e_{t-1} + 0.15) * 0.965, & \text{trigger shielding} \\ (e_{t-1} + 0.8) * 0.965, & 0.9 \le E_{history} \le 1.1 \\ e_{t-1} * 0.965, & \text{otherwise} \end{cases} \tag{16}$$

The second challenge is how to determine the action based on entropy e_t. The change of entropy is reflected in the modification of actors' actions. Due to the use of the deterministic actor-critic framework, actor outputs a tangible action rather than the probability distribution of all actions. We add fluctuations to this action according to the entropy.

$$a_t = H(\pi_\phi(s_t)) = N(\pi_\phi(s_t), 0.2 * e_t) \tag{17}$$

Marten takes the original decision action as the mean and 0.2 times of entropy as the variance. By introducing action exploration mechanism, this model can intelligently adjust the exploration ability of the environment, so as to better adapt to the current network scenario.

V. Implementation

We implement Marten based on Litespeed QUIC (LSQUIC) platform. LSQUIC is an open-source implementation of QUIC

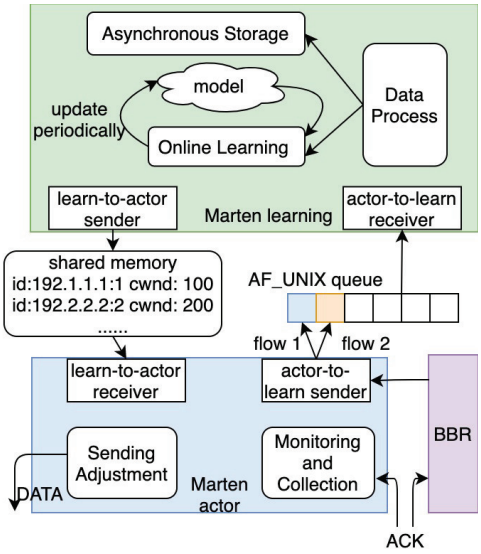


Fig. 5. The implementation of Marten.

and HTTP/3 functionality for servers and clients [26]. Figure 5 shows the overview of the implementation of Marten. The implementation consists of two parts: the Marten actor based on LSQUIC and the Marten learning module based on TensorFlow platform [27]. The Marten actor sends the connection information to the Marten learning module. The BBR component in LSQUIC also sends its cwnd to the learning module to guide the learning process. The learning module in turn configures cwnd and send it to the Marten actor.

BBR uses both pacing rate (determining how quickly to send packets) and cwnd (how many packets can be in flight) to adjust its sending rate. Two parameters are related to the pacing rate and cwnd: $pacing\_gain$ and $cwnd\_gain$. In LSQUIC, $pacing\_rate$ = $pacing\_gain$ * $bottleneck\_bw$, and $cwnd$ = $max(4, cwnd\_gain * bottleneck\_bw * min\_rtt)$. The sender first checks whether $inflight\_packets$ (packets have been sent out in network) exceeds cwnd. If so, no packets will be sent out; otherwise, packets are injected to the network at the rate according to $pacing\_gain$. In the LSQUIC implementation, cwnd does not directly determine the $pacing\_gain$. Instead, it influences the $pacing\_gain$ indirectly through parameters such as $target\_rate$ and $bbr\_max\_bandwidth$, which are network state variables associated with cwnd in the lsquic_bbr structure. Note that, Marten adjusts cwnd, and leaves pacing rate unchanged to enable BBR for bandwidth probe.

Each TCP flow uses the DRL model to obtain its cwnd independently, and is unaware of the action and reward of other flows. All flows' states are sent to the online learning module to update the model. The actor records the ACK information of a connection in real time. In every monitor interval, the actor summarizes the information and sends it to the learning module through socket communication. To reduce the communication overhead between them, we set the monitor interval as n (5 by default) ACK, meaning that after receiving n ACKs, the actor sends summarized information to the learning module. The actor and learning module agree on the communication IP and port number, and AF-UNIX native communication is used for data transmission. In AF-UNIX communication, FIFO queues are used, and the summary information that arrives first will be passed on to the leaning module earlier. The learning module is responsible for regulating the cwnd of each connection.

In order to reduce the communication overhead between the two modules and enable the actor to get cwnd asynchronously, the learning module publishes cwnd for connections to a shared memory. The actor accesses the shared memory every RTT and retrieve the cwnd for each connection. The actor changes the cwnd at next time that it transmits data. We leverage a replacement algorithm based on LRU (Least Recently Used) to manage the shared memory—-when the shared memory is full, the connection information that has not been updated for the longest time is deleted first.

VI. Evaluation

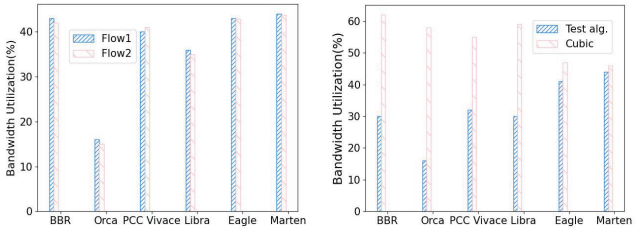
In this section we evaluate the effectiveness of Marten. Evaluations are based on both an emulated environment and production networks.

A. Experimental Setup

The experimental setup can be divided into two parts: simulation environment achieved by OpenAI Gym platform and deployment environment based on LSQUIC platform. There are two kinds of network environments in the deployment: the MahiMahi platform [28] to simulate different kinds of network and production networks over public clouds. For the networks simulated by MahiMahi, we produce the wired network scenarios by setting the parameters (such as minimum RTT, loss rate and bandwidth) at the bottleneck link, and use the cellular traces recorded in the real world to simulate the cellular network. For production networks, we deploy Marten on a public cloud's servers. The server instances are in Asia, North America, and Australia. The clients located in our campus in Shenzhen, China send requests to the servers. The response flows for the requests have different sizes (between 1KB and 50MB).

MI is a particularly fundamental parameter of the hybrid CCAs. In order to receive DRL agent's or BBR's own ACK information in time without interfering with each other, we set the length of MI_t to 1.5*rtt_{t-1}. The weight of the reward function in Marten is set to: ω_1 equals 1, ω_2 equals 0.5, ω_3 equals 10. We will evaluate the performance with different parameter settings later in this section. For DRL model parameters, such as hidden layer size, batch size, and policy network learning rate, we used the default values provided by PyTorch, which are 128, 16, 0.001, respectively.

For baselines, we have implemented representative ML based CCAs Orca [15], Eagle [13], Libra [15], PCC Vivace [29] as well as BBR. Orca is the first deployment hybrid DRL based CC mechanism guided by Cubic, while Eagle is guided by BBR. Libra [15] improves Orca by optimizing the rule-based CCA's guiding stage framework. PCC Vivace is an online learning CCA that uses the online network feedback



(a) using the same algorithm (b) compete with Cubic

Fig. 6. Average bandwidth utilization for simulated wired network.

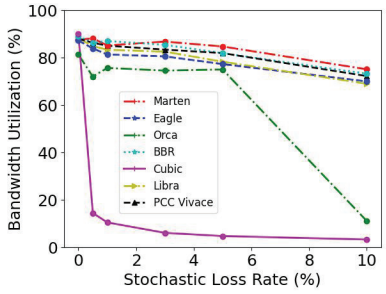
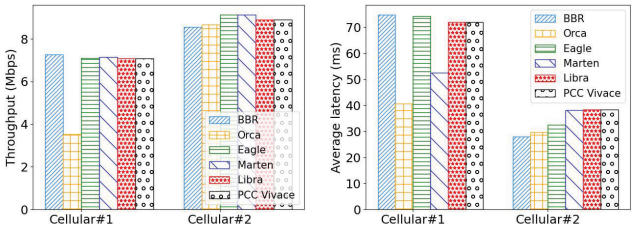


Fig. 7. Performance under different stochastic packet loss rates.



(a) throughput (b) latency

Fig. 8. Adaptability performance under cellular networks.

to guide the sending rate. We implemented these mechanisms and trained respective model by the same data as Marten.Note that since Orca has not open-sourced its learning model, we retrained the model. The training time and data are the same with other learning-based CC mechanisms (Eagle, Libra and Marten). A notable difference between Orca and Libra, Eagle and Marten is that Orca does not use the rule-based CC experience to train its model. Instead, it only uses the rule-based CC to take actions in certain time intervals. In contrast, Libra, Eagle and Marten use the feedback of rule-based CC action to train their models, which accelerate the convergence of the model.

B. Performance Evaluation

1) Performance Evaluation in Emulated Networks: We used Mahimahi to evaluate wired and cellular networks. The wired network's bottleneck bandwidth is 940Mb/s, delay is 30ms (setting by Mahimahi's command). The cellular network is configured by the cellular traces published by TMobile and ATT [28]. In order to better compare the convergence ability of the model, each algorithm model was only trained for 10 hours.

Wired Networks. First, we test two flows using the same algorithm. Figure 6a shows that the bandwidth utilization of two flows of each algorithm is basically the same, but the overall utilization vary greatly. Specifically, the total utilization rates for BBR, Eagle, Libra, PCC Vivace and Marten are 85.23%, 87.82%, 78.91%, 84.32% and 88.78% respectively. In the case of dual-flow competition, Marten's overall utilization ratio is higher than other algorithms.

Second, we fix one flow to use Cubic, and the other flow to use various algorithms. As we can see from Figure 6b, the bandwidth utilization of Cubic is nearly twice that of BBR. However, learning based methods can inhibit such phenomena to a certain extent. For example, the difference between Eagle and Cubic is less than 10%, and the difference between Marten and Cubic is even less than 3%. This further demonstrates Marten's fairness when competing with other algorithms.

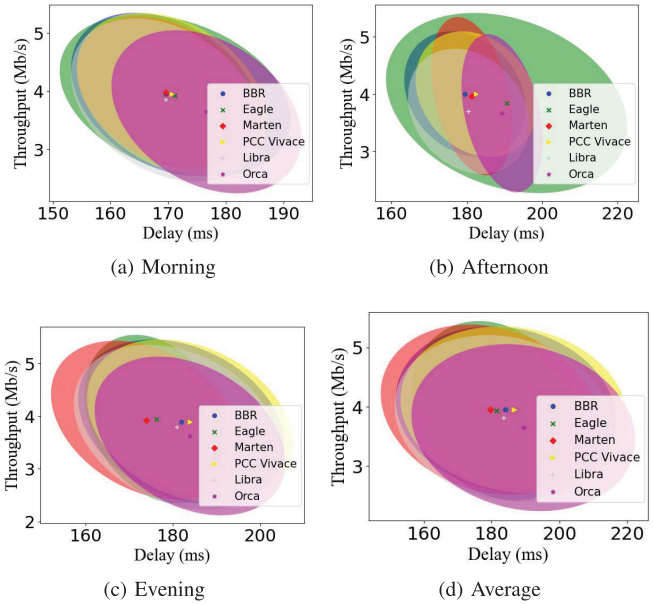
We further change the packet loss rates in wired network. Figure 7 shows the adaptability of rule-based algorithm (BBR / Cubic) and learning-based algorithm (Orca/Libra/PCC Vivace/Eagle/Marten) to different stochastic packet loss rates. BBR controls the transmission rate by bandwidth delay product (BDP), so it is insensitive to packet loss. Cubic regards packet loss as the signal of congestion, so it is very sensitive to random packet loss. Learning-based algorithms can effectively deal with the disturbance caused by random packet loss because of online learning model. Note that as Orca uses Cubic as the expert, its performance drops quickly when the loss rate exceeds 5%.

Cellular Networks. Figure 8 shows that in cellular network, Marten is always in the first echelon. Compared with Eagle, Marten improves the bandwidth by 0.39%, reduces the latency by 21.24%. Compared with BBR, Marten improves the bandwidth by 4.18%, reduces the latency by 27.37%. While Marten's throughput is close to that of Libra and PCC Vivace, it reduces the latency by 14.6% and 15.67% on average respectively. We can also see that Eagle and BBR have similar behaviors. This phenomenon shows that the Eagle model is largelyy affected and limited by BBR in the training process. Although Marten also uses BBR as the expert, it improves the performance of BBR in the cellular#2 scenario.

2) Performance In-the-Wild: In addition to using Mahimahi for real-world trace reproduction, we deployed and tested the Marten framework on cloud production networks.

We used Alibaba Cloud servers and set up inter-continent communication experimental scenarios. We choosed Silicon Valley in the Americas, Zhangjiakou in Asia, and Sydney in Australia for inter-continent communication. We tested BBR, Ocra, Libra, PCC Vivace, Eagle, and Marten in three time periods: morning, afternoon, and evening, and obtained the overall performance after summarizing the information. The experimental testing is divided into two parts. The first part is a single flow without bandwidth competition, and the second part is a multi-flow test with competition.

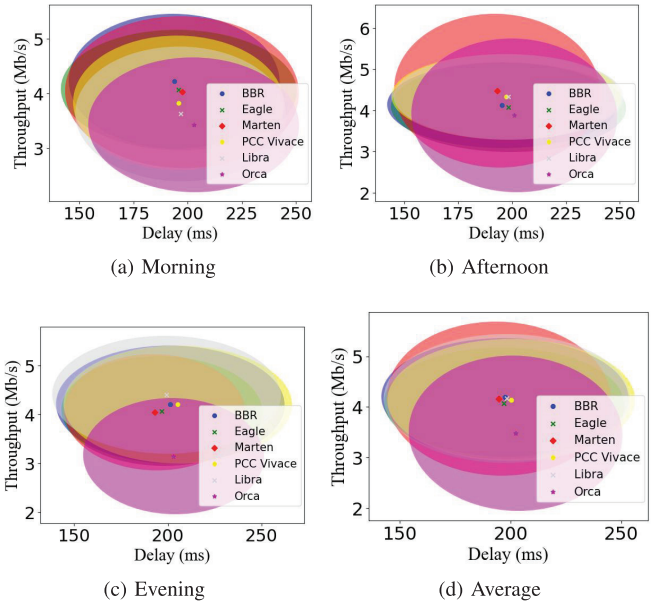
Single flow. First, we focus on the performance of each algorithm on single flow in Figure 9. Marten exhibits lower latency in data transmission in all time periods. In terms of overall performance (shown in Figure 9d), while the throughput of different solutions is close, Marten has lower latency,



(a) Morning (b) Afternoon

(c) Evening (d) Average

Fig. 9. Single flow for inter-continental communication.



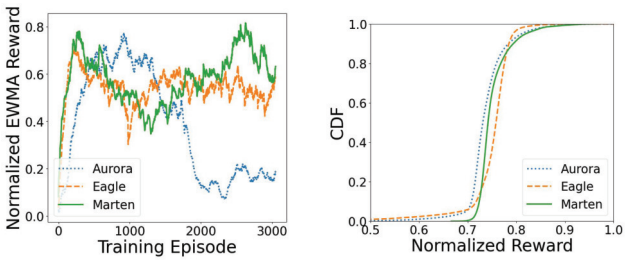
(a) Morning (b) Afternoon

(c) Evening (d) Average

Fig. 10. Multi-flows for inter-continental communication.

which is 2.55% lower than BBR, 1.09% lower than Eagle, 2.21% lower than Libra, 4.34% lower than PCC Vivace, and 6.34% lower than Orca.

Multi-flows. Figure 10 shows that Marten achieves the top performance for throughput or latency in all the scenarios. For example, Marten has a large bandwidth improvement during the afternoon period, with a bandwidth utilization improvement of 7.86% compared to BBR, 9.03% compared to Eagle, 3.53% compared with PCC Vivace. The overall performance shown in Figure 10d reveals that Marten improves throughput by 2.23% and reduced latency by 1.18% compared to Eagle. Compared with Libra and BBR, Marten reduces latency by 3.45% and 1.37% respectively. The reason that Orca performs poorer than others lies in the fact that Orca does not use the



(a) EWMA (b) CDF

Fig. 11. Comparison of the CDF and EWMA reward value.

rule-based CC experience to train its model. Instead, it only uses the rule-based CC to take actions in certain time intervals. This reveals the importance of the rule-based CC experience during training.

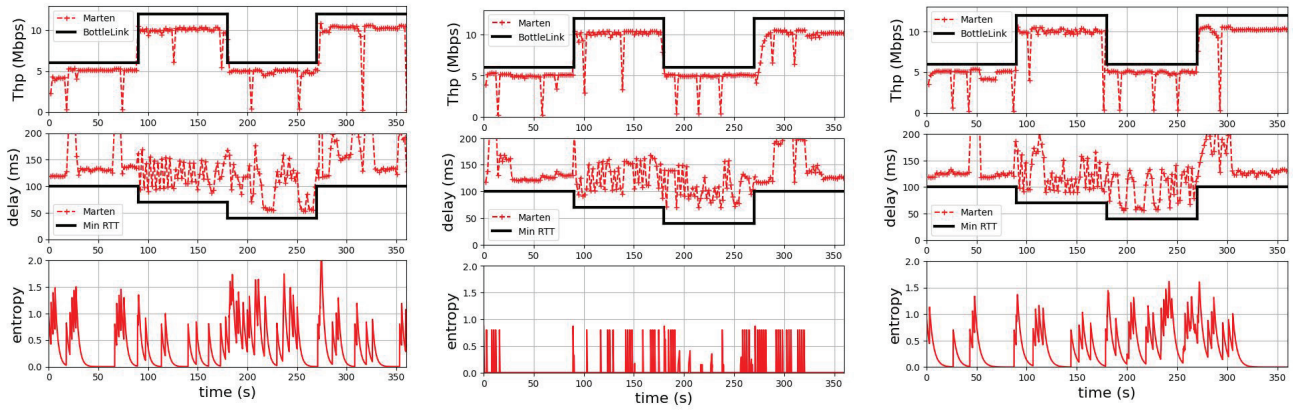
C. Validating Safety Guide

In this subsection, we validate that Marten eliminates long tail wrong actions by safety guide. Compared with Aurora, Marten has more advantages than simply using rule-based or learning-based methods by combining rule-based algorithm and learning-based algorithm.

We conduct repeated tests several times. Figure 11a shows the normalized EWMA mean of clean-slate Aurora, Eagle and Marten with the training episodes. Marten and Eagle with expert guidance have the characteristics of rapid convergence compared with Aurora, and both can keep the reward at a high value. Marten's EWMA reward is higher than both Eagle and Aurora, even twice that of Aurora. With the help of the expert guidance of BBR in Marten, shielding mechanism steers the system to advance towards a better model with high generalization ability, which can make the algorithm adapt to current environments faster.

Figure 11b shows that Marten has improved the top and bottom reward value at the same time. Marten outperforms Aurora and Eagle at top 20%. Aurora can explore the whole action space at will, but it needs to bear the consequences of making errors. Therefore, the convergence speed of Aurora is slow, and it cannot adapt quickly in the process of scenario switching. Eagle is limited by expert algorithms, and always shrinks in the safe state that experts think. However, this leads to the limited exploration of action space, leading to performance degradation [30]. In contrast, Marten does not completely refuse the existence of unsafe actions, but with the help of shielding mechanism, the model can quickly recover from unsafe actions without disastrous consequences. In this process, Marten explores some actions beyond the safe state, which are beneficial to the model's adaptation to different scenario.

Meanwhile, Marten also effectively alleviates the long tail phenomenon caused by the reinforcement learning exploration mechanism. This phenomenon limits the performance of the transport layer and the application layer. Table IV lists the quantiles of the three algorithms. By comparison, Marten has higher values in all kinds of quantiles and on average.



(a) $e_t = \begin{cases} (e_{t-1} + 0.15) * 0.965, & \text{shielding} \\ (e_{t-1} + 0.8) * 0.965, & \text{remain safe} \\ e_{t-1} * 0.965, & \text{otherwise} \end{cases}$ (b) $e_t = \begin{cases} (e_{t-1} + 0.15) - 0.13, & \text{shielding} \\ (e_{t-1} + 0.8) - 0.13, & \text{remain safe} \\ e_{t-1} - 0.13, & \text{otherwise} \end{cases}$ (c) $e_t = \begin{cases} (e_{t-1} + 0.1) * 0.975, & \text{shielding} \\ (e_{t-1} + 0.7) * 0.975, & \text{remain safe} \\ e_{t-1} * 0.975, & \text{otherwise} \end{cases}$

Fig. 12. Discussion for Marten's entropy parameters choice.

TABLE IV
QUANTILE COMPARISON OF TOTAL NORMALIZED REWARD

reward	**aver.**	95%	90%	80%	10%	5%
Aurora	0.74	0.81	0.78	0.76	0.71	0.70
Eagle	0.75	0.79	0.78	0.77	0.72	0.69
Marten	**0.75**	**0.82**	**0.80**	**0.77**	**0.72**	**0.72**

TABLE V
PERFORMANCE FOR ONLINE AND OFFLINE LEARNING

Thr(Mbps)	**Wired**	**Cellular**
online	**4.89**	**12.30**
offline	3.65	7.69

TABLE VI
PERFORMANCE FOR DIFFERENT ENTROPY STRATEGY

schemes	**link utilization**	**average latency(ms)**
a	**80.45%**	133.31
b	79.04%	132.18
c	79.94%	**129.31**

D. Online Learning Effectiveness

Using online learning model is more suitable for scene changes than offline learning model. We train the model in OpenAI Gym. In order to verify that the obtained model still needs online learning to adapt to the network in LSQUIC, we conducted a set of comparative experiments. The network is simulated by Mahimahi (the same wired and cellular network as in Section VI-B).

In this set of experiments, we use the exploration mechanism based on entropy to adjust the exploration degree of the model in real time, and train the model after the model gets a round of reward. In the offline learning test, we use the model trained in OpenAI Gym to infer the action, and the model parameters will not change with the change of the network. Table V shows that online learning improves the throughput by 20% in the wired network and 1.5 times the original throughput in the cellular network. Therefore, online learning plays an important role in reinforcement learning in congestion control.

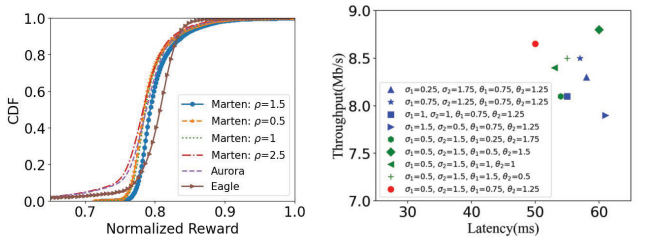
E. Parameters Choice Evaluation

Figure 12 shows the change of Marten's decision with three groups of entropy parameters during the continuous switching of network bottleneck links and minimum RTT. The maximum bottleneck link and the minimum RTT of these four scenarios are identified by black solid lines.

Due to the exploratory nature of DRL algorithm, DRL sometimes performs poorly. When the DRL model makes wrong actions, such as the 25s and 70s periods in Scheme *a*, the throughput decreases significantly and the RTT increases significantly. At this time, the expert shielding mechanism is triggered, and the entropy increases continuously. In the third scenario of Scheme *a*, the entropy value increases jitter, because Marten detects the possibility of smaller RTT and tries to find a better control scheme through entropy adjustment. This phenomenon also exists in the third scenario of Scheme *c*. We can conclude that when the throughput is lower than bottleneck and the delay is high than minimum RTT, the entropy will continue to increase. When the DRL model acts as expected, the entropy will naturally decay, so that the model can gradually converge.

In the three schemes, Scheme *a* and Scheme *c* use addition growth and division attenuation, while Scheme *b* uses addition and subtraction growth and attenuation. Through this set of experiments, we find that the selection of scheme *a* is the best. Scheme *b* adopts subtraction attenuation, which makes the adjustment of entropy not smooth. The performance of scheme *c* is also not as good as that of scheme *a*.

We further discuss the value of ρ, σ_1, σ_2, θ_1 and θ_2 in Eq. 10, Eq. 11 and Eq. 12 respectively. Recall that we use the change of ρ*RTT to decide the unsafe state caused by the



(a) Reward value for ρ (b) Performance for σ_1,σ_2,θ_1, and θ_2

Fig. 13. Performance for different value of parameters.

DRL's wrong actions. We evaluate the reward for different values of ρ in Figure 13a. It shows that when $\rho = 2.5$, Marten's reward value is close to Aurora as Marten seldom marks the state is unsafe0. On the other hand, setting $\rho = 0.5$ or 1 is too aggressive and may trigger experts overly). Setting $\rho = 1.5$ achieves the best balance.

The parameters σ_1, σ_2, θ_1 and θ_2 decide the weight of reward value stored in replay memory for online learning when Marten triggers shielding mechanism. Large σ_1, θ_1 and small σ_2, θ_2 mean the reward values have higher weights, otherwise they have lower weights. Figure 13b shows the performance for different parameter settings. For larger σ_1, θ_1, Marten has longer latency and smaller throughput as the it treats the guide from expert too important. We find that when $\sigma_1 = 0.5$, $\sigma_2 = 1.5$, $\theta_1 = 0.75$ and $\theta_2 = 1.25$, Marten achieves the best performance. So by default, we set $\sigma_1 = 0.5$, $\sigma_2 = 1.5$, $\theta_1 = 0.75$ and $\theta_2 = 1.25$. Note that the parameter settings discussed above are all driven by our experiments. They can be tuned to fit other experiments by offline evaluation as in our paper before deployment in the wild.

F. Fairness and Friendliness

In this subsection, we further elaborate on the fairness issue of Marten. References [31] and [32] introduce the Jain's fairness index and use this value to evaluate the fairness of each stream under different algorithms. The calculation method for this fairness index is:

$$fairness = \left(\sum_{i=1}^{n} thr_i\right)^2 \Big/ \left(n * \left(\sum_{i=1}^{n} (thr_i)^2\right)\right) \quad (18)$$

The i in equation represents flow i, thr_i represents the throughput of the $i-th$ flow, n is the total number of flows. The range of this fairness index is between 0 and 1. When all flows allocate bandwidth fairly, the fairness index will reach its maximum value.

We select fragments from the inter-continent data and plot the fairness index curves in Figure 14. The two flows use the same algorithm while competing for the same link with a bottleneck bandwidth of 4Mbps. We use blue and red lines to plot their throughput over time, using the y-axis on the left as the coordinate axis. The top black line is the Jain's fairness index curve, using the y-axis on the right as the coordinate axis.

For BBR, it can be seen that during the initial startup, flow 1 (blue line) quickly occupies the bandwidth resources in the bottleneck link, while flow 2 (red line) is forced to end the fast startup process early. This unfair phenomenon is gradually resolved in the process of bandwidth throughput detection and minimum RTT detection of BBR, and high fairness is maintained in the future. For Eagle, there are two significant instances of unfairness at 200 and 400 seconds, with flow 1 taking over the throughput of user 2. The Jain's fairness index decreases to 0.9 at one point and is fixed within approximately 20-40 seconds. For Marten, there is a certain degree of unfairness at 310 seconds, and the Jain's fairness index decreases to 0.97, while the overall fairness remains sufficient for the rest of the time. The main reason why Marten has better fairness compared to Eagle is that Marten has added trend prediction to detect DRL action's error and shielding mechanisms, allowing flows to quickly correct their aggressive actions. At the same time, the dynamic exploration mechanism of entropy also makes the Marten model converges to a more optimal solution faster.

G. Performance Using Different Reward Functions

Considering the needs of different applications, in this subsection we conduct additional discussions on the types and parameters of reward functions. Software download usually focus more on high throughput, video live streaming pursues low latency, while video on demand pursues high image quality and low lag, and online games pursue even zero latency. Various types of applications have different requirements for throughput and latency. By using a dynamic exploration mechanism and a safety learning algorithm based on shielding mechanism, Marten can converge to a better solution faster. Meanwhile, Marten can also achieve tendency towards different needs through online learning and modifying parameters in the reward function. We select the reward functions of Orca [14] and Libra [15] for comparison.

We first introduce the three types of reward functions. The first kind of reward function is the linear function used by Marten as show by Eq. 2. The second kind is Libra's modification of linear function (shown in EQ 19), which divides the reward value of the current round from the previous round, and uses the difference as the reward value for training model.

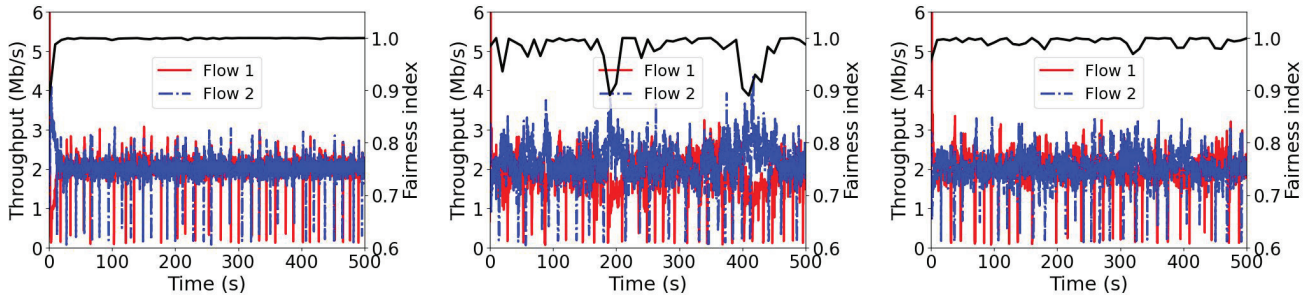
$$R = \Delta R_t = R_t - R_{t-1} \quad (19)$$

where R_t is a linear equation with throughput, latency and loss rate [15].

The third type of reward function (shown in Eq. 20 and Eq. 21) used by Orca followed the principle of *Power* [33]. *Power* is a well-studied metric in the context of congestion control and is defined as $Power = throughput/delay$

$$\hat{R} = R/R_{max} = \left(\frac{throughput - \eta * loss}{delay'}\right) \Big/ \left(\frac{thr_{max}}{d_{min}}\right) \quad (20)$$

$$delay' = \begin{cases} d_{min} & (d_{min} \le delay \le *d_{min}) \\ delay & o.w. \end{cases} \quad (21)$$



(a) Fairness of BBR (b) Fairness of Eagle (c) Fairness of Marten

Fig. 14. Fairness for multi-flows of different algorithms.

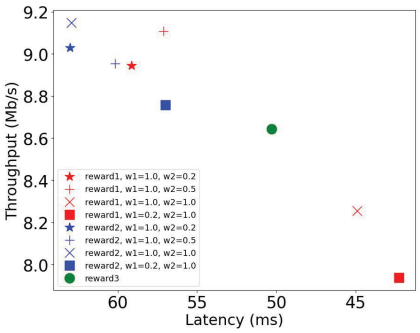


Fig. 15. Performance comparison of different types of reward functions and parameters.

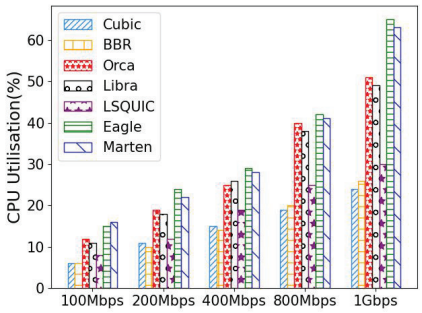


Fig. 16. Computational overhead for different CCAs.

As the first two types of reward functions can emphasize throughput and latency by setting different weights on these two metrics, we conduct experiments with multiple sets of parameter settings. Specifically, let ω_1 denote the weight of throughput, and ω_2 is the weight of latency. Changing the weights in the reward function can guide the model to approach the designer's desired preference. We set four sets of parameters, namely $\omega_1 = 1.0$, $\omega_2 = 0.2$; $\omega_1 = 1.0$, $\omega_2 = 0.5$; $\omega_1 = 1.0$, $\omega_2 = 1.0$; $\omega_1 = 0.2$, $\omega_2 = 1.0$. The third type of reward function cannot be parameterized as it is a multiplication and division method. The experimental scenario is a cellular network scenario simulated by the MahiMahi. Figure 15 shows the comparison results.

For the first type of reward function, when $\omega_1 = 0.2$ and $\omega_2 = 1.0$, the model achieves lower latency at the cost of reducing throughput (the red square point at Figure 15). When $\omega_1 = 1.0$ and $\omega_2 = 0.5$, the model meets high throughput requirements, with an average throughput of 9.11Mb/s (the red cross point at Figure 15). These results effectively demonstrate the ability of meeting different preferences by the first two types of reward functions.

We can see the similar effects for the second type of reward function. it can be seen that the modification of the coefficient also brings some changes. With the same settings $\omega_1 = 0.2$ and $\omega_2 = 1.0$, the first type of reward function achieves a latency 25.85% lower than that of the second type (which is 57.01ms, the blue square point at Figure 15). The reason is that the second type of reward function uses the difference between the reward values of two rounds as the reward value, but this difference cannot reflect the desired preferences.

The third type of reward function adopts a multiplication and division method. It achieves moderately high latency and throughput. But it cannot support the implementation of different preferences.

In summary, the reward function used in Marten can reflect the application preferences and achieve better performance.

H. Overhead Analysis

We finally evaluate the computation overhead of different CCAs by comparing their CPU overhead. We change the bottleneck bandwidth from 100Mbps to 1Gbps in Mahimahi. The delay is 30ms and the buffer is 1 BDP. The servers are with two Intel(R) Xeon(R) Silver 4208 CPUs, 16 CPU cores, 64GB memory and 10Gb/s NIC. We increase the number of requests from the client until the bottleneck is saturated. We use the the Linux system top command to obtain the CPU utilization under different bandwidth. Seven CCAs are compared, including four Linux kenel CCAs: Cubic, BBR, Orca and Libra; three user-space CCAs: vanilla LSQUIC, Eagel and Marten (Eagel and Marten are both based on LSQUIC).

Figure 16 reports the results. As expected, all CCAs' CPU utilizations increase as the traffic increases. Under the same bandwidth, the utilization of kernel CCAs is lower than the LSQUIC based CCAs. For the kernel CCAs, Orca and Libra occupy more CPU than Cubic and BBR caused by the learning process. For the LSQUIC based CCAs, Eagel and Marten's utilization is comparable; their CPU usage is also larger than

the vanilla LSQUIC. While such CPU overhead is inevitably for learning-based CCAs, reducing the computational overhead caused by learning process worths further investigation.

VII. Discussion

Intelligently adjust the value of entropy. In Marten, we hard-code the value of increase and decrease of entropy—the model modifies the exploration degree according to the artificially set entropy change rules. From the experimental results, this approach is effective to dynamically adjust the exploration ability of the model according to the changes of the environment. However, there is still some room for improvement. In some cases, the relationship between the ability of model exploration and the safe state of model destruction is unknown. Therefore, there is a trace-off between flexible adaptation (fast entropy increase) and decision convergence (entropy annealing) of the model. How to balance the relationship between the two is still a big challenge.

Definition of safety state. Many practical problems have clear safety boundaries, but in congestion control, the safety boundary is fuzzy. In dynamic networks, Marten uses latency-delay based filters to filter part of the noise and tries to distinguish between network jitter and model errors. More accurately identifying the wrong actions of the model will further improve the significance and value of safety learning.

VIII. Related Work

Rule based CC. There are many rule-based CC mechanisms that are specifically designed for different networks. Cubic [1], BBR [2] and Copa [5] are designed for WAN. C2TCP [3], Sprout [8] are designed for cellar network, while DCTCP [4], TIMELY [6] are for DCN. Marten uses the rule based CC as an expert to guide the DRL actions.

Machine learning on CC. RemyCC [34] is the first CC mechanism which uses DecPOMDP [35] based distributed algorithm to determine the sending rate. PCC-Vivace [29] determines the sending rate according to the network feedback by online learning method. DeepCC [12], AUTO [11], Aurora [9] and MOCC [10] are all DRL based CC mechanisms which have different optimization targets. These pure DRL based CC mechanisms suffer from the shortcoming of DRL such as long tail effect. For example, AUTO is designed specifically for satellite network while MOCC aim to achieve different users' objectives. Eagle [13], Orca [14] and Libra [15] use the rule based CC mechanism (Cubic or BBR) as an expert to overcome the limitation of the DRL. Marten is a supplement for these mechanisms by polishing the expert to expand the exploration of DRL's actions.

Safe Learning. Safe learning can guarantee that RL algorithm make the right action. Online safe RL [36] uses known-safe fallback policy to determine its actions if the system enters an unsafe region. Mnih et al. [37] introduce entropy to measure the exploration of the objective function, and then explores greedy if the entropy is large. Xie et al. [38] use controller to guide the training process and that can speed up the training. Rong and Luan [39] introduce multiple training levels to keep the safety of action policy. Marten follows these safe learning algorithms and extends them to CC mechanism.

Optimization and implementation for QUIC. QUIC is a multiplexed transport mechanism which implements its CC logic in the user space while transferring data by the UDP. As most of its control logic is in user space, it is easier to implement optimization for control logic than in kernel and that promotes many optimization in QUIC. Xlink [40] achieves QoE driven multi-path transport in QUIC for video services. To extend the TCP protocol function, PQUIC [41] enables QUIC clients and servers to dynamically exchange protocol plugins in QUIC. MIMIQ [42] masks the user's IP to protect user identity by leveraging QUIC's connection migration capability. Yang et al. [43] increase the QUIC's effectiveness by offloading the QUIC packet to the programable NIC. Marten is one of the optimizations for QUIC, and is the first DRL based CC implemented in QUIC.

IX. Conclusion

In this paper, we propose Marten, a DRL-based CCA using entropy exploration mechanism and expert guided safety mechanism. Marten combines the advantages of rule-based algorithm and learning-based algorithm, and can adjust the model in real time according to the changes of the network to achieve rapid convergence. We implement Marten in LSQUIC, a production open source version of QUIC. The experiments both in OpenAI Gym and LSQUIC show Marten's advantages over BBR, Orca and Eagle.

References

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

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[3] S. Abbasloo, Y. Xu, and H. J. Chao, "C2TCP: A flexible cellular TCP to meet stringent delay requirements," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 4, pp. 918–932, Apr. 2019.

[4] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, Aug. 2010, pp. 63–74.

[5] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2018, pp. 329–342.

[6] R. Mittal et al., "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.

[7] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

[8] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. 10th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2013, pp. 459–471.

[9] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3050–3059.

[10] Y. Ma et al., "Multi-objective congestion control," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 218–235.

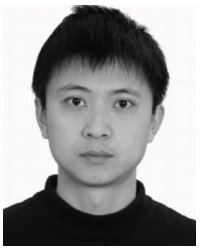
[11] X. Li, F. Tang, J. Liu, L. T. Yang, L. Fu, and L. Chen, "AUTO: Adaptive congestion control based on multi-objective reinforcement learning for the satellite-ground integrated network," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 611–624. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/li-xu

[12] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna make your TCP scheme great for cellular networks? Let machines do it for you!," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 265–279, Jan. 2021.

[13] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 676–685.
[14] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 632–647.
[15] Z. Du, J. Zheng, H. Yu, L. Kong, and G. Chen, "A unified congestion control framework for diverse application preferences and network conditions," in *Proc. 17th Int. Conf. Emerg. Netw. Exp. Technol.* New York, NY, USA, Dec. 2021, pp. 282–296.
[16] G. Brockman et al., "OpenAI gym," 2016, *arXiv:1606.01540*.
[17] A. Langley et al., "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 183–196.
[18] M. Bishop, *HTTP/3*, document RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114
[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[20] C. Tessler et al., "Reinforcement learning for datacenter congestion control," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 49, no. 2, pp. 43–46, Jun. 2022.
[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 1889–1897.
[22] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
[23] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 387–395.
[25] S. Scherrer, M. Legner, A. Perrig, and S. Schmid, "Model-based insights on the performance, fairness, and stability of BBR," in *Proc. 22nd ACM Internet Meas. Conf.*, Oct. 2022, pp. 519–537.
[26] [Online]. Available: https://github.com/litespeedtech/lsquic
[27] [Online]. Available: https://github.com/tensorflow/tensorflow
[28] R. Netravali et al., "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 417–429.
[29] M. Dong et al., "PCC vivace: Online-learning congestion control," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement.*, 2018, pp. 343–356.
[30] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2018, vol. 32, no. 1, pp. 1–13.
[31] S. Floyd, *Metrics for the Evaluation of Congestion Control Mechanisms*, document RFC 5166, 2008.
[32] F. Yang, Q. Wu, Z. Li, Y. Liu, G. Pau, and G. Xie, "BBRv2+: Towards balancing aggressiveness and fairness with delay-based bandwidth probing," *Comput. Netw.*, vol. 206, Apr. 2022, Art. no. 108789.
[33] A. Giessler, J. Hänle, A. König, and E. Pade, "Free buffer allocation—An investigation by simulation," *Comput. Netw.*, vol. 2, no. 3, pp. 191–208, Jul. 1978.
[34] 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.
[35] F. A. Oliehoek, "Decentralized pomdps," in *Reinforcement Learning*. Cham, Switzerland: Springer, 2012, pp. 471–503.
[36] H. Mao, M. Schwarzkopf, H. He, and M. Alizadeh, "Towards safe online reinforcement learning in computer systems," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, 2019, pp. 1–16.
[37] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[38] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 6276–6283.
[39] J. Rong and N. Luan, "Safe reinforcement learning with policy-guided planning for autonomous driving," in *Proc. IEEE Int. Conf. Mechatronics Autom. (ICMA)*, Oct. 2020, pp. 320–326.
[40] Z. Zheng et al., "XLINK: QoE-driven multi-path QUIC transport in large-scale video services," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 418–432.
[41] Q. De Coninck et al., "Pluginizing QUIC," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 59–74.
[42] Y. Govil, L. Wang, and J. Rexford, "MIMIQ: Masking IPs with migration in QUIC," in *Proc. 10th USENIX Workshop Free Open Commun. Internet*, Jan. 2020, pp. 1–16. [Online]. Available: https://www.usenix.org/conference/foci20/presentation/govil
[43] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC quicker with NIC offload," in *Proc. Workshop Evol., Perform., Interoperability QUIC*, Aug. 2020, pp. 21–27.

Jianer Zhou received the Ph.D. degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), in 2016. He is currently an Associate Professor with the Pengcheng Laboratory, Shenzhen, China. His research interests lie in network performance, future internet architecture, and internet measurement.

Zhiyuan Pan received the Master of Engineering degree from Southern University of Science and Technology in 2024. He is currently an Engineer at Tencent, Shenzhen, China. His research interests lie in network performance and internet measurement.



Zhenyu Li received the B.S. degree from Nankai University in 2003 and the Ph.D. degree from the Graduate School, Chinese Academy of Sciences, in 2009. He is a Professor at the Institute of Computing Technology, Chinese Academy Sciences. His research interests include internet measurement and networked systems.

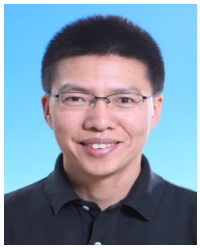
Gareth Tyson received the Ph.D. degree from Lancaster University in 2010. He is an Assistant Professor at The Hong Kong University of Science and Technology. His research interests include internet measurements, web computing, and content distribution.

Weichao Li received the B.E. and M.E. degrees from South China University of Technology, Guangzhou, China, in 2002 and 2009, respectively, and the Ph.D. degree from the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, in 2017. He is currently an Associate Professor at the Peng Cheng Laboratory, Shenzhen, China. His current research interests include network measurement, especially on internet measurement technologies, large-scale network performance monitoring and diagnosis, and mobile network performance monitoring, SDN, industrial internet, cloud computing, quality of experience (QoE) measurement, and machine learning.

Xinyi Qiu received the Master of Engineering degree from the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), in 2018. She is currently an Engineer at the Department of New Networks, Peng Cheng Laboratory, Shenzhen, China. Her research interests lie in network performance, future internet architecture, and internet measurement.

Heng Pan received the Ph.D. degree in computer science from the University of Chinese Academy Sciences in 2018. He is an Associate Professor at the Computer Network Information Center, Chinese Academy Sciences. His research interests include SDN/NFV, distributed systems, and in-network computation.

Xinyi Zhang received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2022. She is currently an Associate Professor at the Computer Network Information Center, Chinese Academy of Sciences. Her research interests include internet architecture and internet measurement.



Gaogang Xie received the Ph.D. degree in computer science from Hunan University in 2002. He is currently a Professor at the Computer Network Information Center, Chinese Academy of Sciences, and the University of Chinese Academy of Sciences. His research interests include internet architecture, packet processing and forwarding, and internet measurement.