A Multi-objective Reinforcement Learning Perspective on Internet Congestion Control

Zhenchang Xia*, Yanjiao Chen†§, Libing Wu*§, Yu-Cheng Chou*, Zhicong Zheng*, Haoyang Li* and Baochun Li‡
*Wuhan University, P. R. China
†Zhejiang University, P. R. China
‡University of Toronto, Canada
§Co-corresponding author

***Abstract*—The advent of new network architectures has resulted in the rise of network applications with different network performance requirements: live video streaming applications require low latency. In contrast, file transfer applications require high throughput. Existing congestion control protocols may fail to simultaneously meet the performance requirements of these different types of applications since their designed objective function is fixed and difficult to readjust according to the needs of the application. In this paper, we develop MOCC (Multi-Objective Congestion Control), a novel multi-objective congestion control protocol that can meet the performance requirements of different applications without the need to redesign the objective function. MOCC leverages multi-objective reinforcement learning with preferences in order to adapt to different types of applications. By addressing challenges such as slow convergence speed and the difficulty of designing the end of the episode, MOCC can quickly converge to the equilibrium point and adapt multi-objective reinforcement learning to congestion control. Through an extensive array of experiments, we discover that MOCC outperforms the most recent state-of-the-art congestion control protocols and can achieve a trade-off between throughput, latency, and packet loss, meeting the performance requirements of different types of applications by setting preferences.**

***Index Terms*—congestion control, multi-objective reinforcement learning, communication network, TCP initial window**

I. Introduction

Multi-objective reinforcement learning has gained widespread popularity due to its applicability to solving problems with multiple conflicting objectives. Compared to traditional congestion control protocols, where the designed objective function is fixed, the optimal policy in a multi-objective congestion control protocol relies on the application's performance requirements. For example, there are numerous applications on smartphones (Fig 1), including latency-sensitive and throughput-sensitive applications. Depending on the type of application and its network performance requirements (e.g. high throughput, low latency, and low packet loss), the agent might need to follow completely different policies. If an application is a throughput-sensitive file transfer application and throughput is critical, the application requires high throughput for file transfer, will prefer congestion control protocols that improve higher throughput and will have relatively low latency requirements. Moreover, the agent's reward function might reward the increase of throughput. On the other hand, if an application is a latency-sensitive live streaming application for which minimizing delay is crucial, it requires low transmission latency to reduce video jams and will have relatively low packet loss requirements. The agent's reward function in this situation needs to reward delay reduction. Therefore, using the same objective function for applications with different performance preferences will limit the congestion control algorithm's ability to improve the network application's performance.

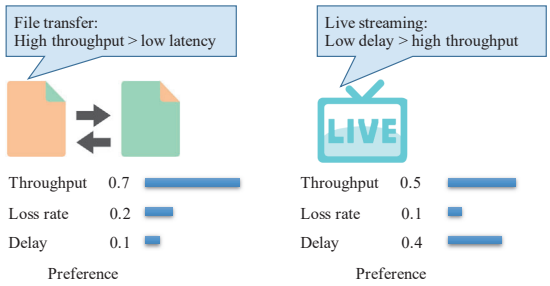


Fig. 1: Performance-oriented applications are a real-life example of different preferences.

Over the past decades, extensive active research has been conducted on Congestion Control (CC) with the goal of achieving higher throughput, lower delay, and lower loss rates for network services such as file transfer, video streaming, and online gaming. Traditional congestion control protocols aim to reduce network congestion according to predetermined rules. For example, the delay-based congestion control protocols, Verus [1], Vegas [2], and LEDBAT [3] use delay changes as congestion control signals and adjust the congestion window using the measured delay. However, while such delay-based protocols can lower latency, they cannot improve network throughput, and they can only meet the performance needs of delay-sensitive applications. Several recently proposed congestion control protocols, such as Copa [4], and BBR [5], attempt to achieve a trade-off between delay and throughput; unfortunately, however, these approaches can cause degradation of both performance metrics.

Recently, learning-based congestion control protocols have become enormously popular. Unlike traditional congestion control protocols, learning-based protocols control congestion using a real-time network state rather than pre-defined rules. Unlike traditional algorithms, PCC [6] and PCC Vivace [7] learn the relationship between sending rate control and

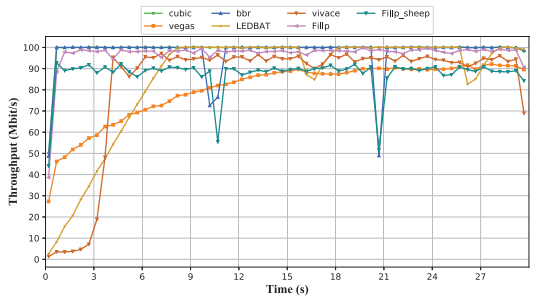


Fig. 2: The steady-state behavior

performance in an online fashion. Unfortunately, the main limitation is the low convergence speed in the congestion control protocol for online learning. Indigo [8] employs offline learning to determine sending rates by using a recurrent neural network (RNN) to store the mapping from states to actions. As the latest congestion control protocol, Aurora [9] uses deep reinforcement learning algorithms to control the sending rate. However, the objective function or reward function of these approaches is fixed, making it impossible for them to meet the performance needs of more than one type of application, e.g., delay-tolerant network [10], smartphone applications [11] or real-time streaming services.

Accordingly, to meet the performance needs of different types of applications, we propose a novel congestion control algorithm, called Multi-Objective Congestion Control (MOCC), which uses multi-objective reinforcement learning for performance goals for different types of applications. To begin with, MOCC trains a single policy network that is optimized over the entire space of preferences in congestion control. This enables the trained model to produce the optimal policy for any given preference; unlike existing protocols, the objective function is fixed, making it challenging for these protocols to meet the needs of a wide range of applications. In addition, we design a novel dynamic initialization method to set the initial value of the congestion window depending on the network bandwidth, thus efficiently accelerating the convergence. Furthermore, the MOCC agent can end an episode in the most appropriate manner by designing a terminal algorithm that draws on the concepts of a winning game, losing the game, and tie-breaker derived from game theory.

To test the performance of MOCC, we conduct extensive experiments using the Pantheon platform and Linux kernel in which we compare MOCC with a large number of state-of-the-art congestion control algorithms. MOCC achieves a trade-off between high throughput and low latency and accordingly outperforms recent state-of-the-art congestion control protocols in different emulated networks at 12Mbps and 50Mbps. For different cellular network scenarios, MOCC is found to meet the performance needs for different types of applications and exceed the performance of other protocols by setting different preferences.

II. Motivations and Challenges

In recent years, a number of new congestion control protocols have been proposed to solve network congestion problems and improve network performance. However, these new congestion control protocols have introduced their own problems. In the following, we will summarise the problems with the existing protocols and outline the key challenges that MOCC needs to address.

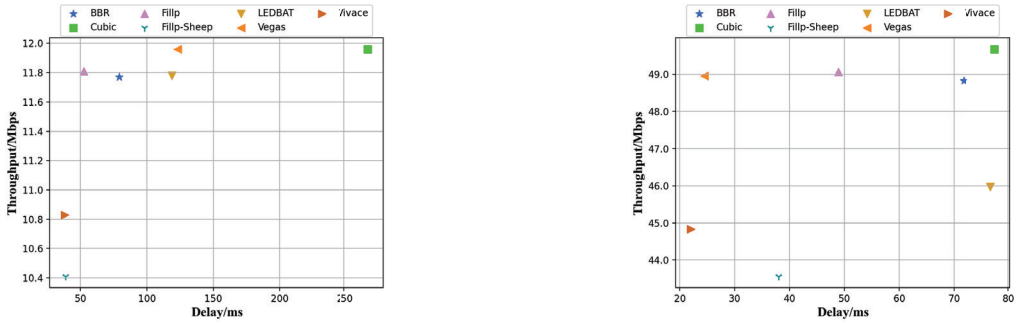
A. Motivations

To shed light on the problems with current congestion control protocols, we perform experiments to examine the existing protocols' convergence behavior on links with 100Mbps bandwidth in Fig 2. We further test the different trade-off points between latency and throughput for common protocols on links with a bandwidth of 12Mbps and 50Mbps and present the results in Fig 3. We test both rule-based schemes and classic schemes, including include TCP Cubic [12], Vegas [2], BBR [5], LEDBAT [3], PCC Vivace [7], Fillp [8], Fillp-Sheep [8] to verify their performance. After analyzing the results of these experiments, we identify two key performance issues with current congestion control protocols:

- **Convergence issue.** We test the common protocols' change in throughput over time to assess their convergence behavior (see Fig 2). From the figure, we can observe that the learning-based congestion control protocol, PCC Vivace, needs more time to converge to a stable condition. This is because learning-based congestion control protocols use prior knowledge to train the network model without defining the proper initialization congestion window (init-cwnd) to improve the network convergence speed according to the network conditions. Moreover, it is also difficult to improve convergence speed in a shorter time when the initialization window of the network is relatively small (similar to LEDBAT in Fig 2).
- **Inability to achieve a trade-off between different performance metrics.** In Fig 3, we test the different trade-off points between latency and throughput for common protocols to determine whether these can be balanced by existing protocols. We can observe that the latency of the protocol varies significantly in different network environments (e.g. TCP Vegas, LEDBAT, TCP Cubic). For example, TCP Cubic has a lower delay at 50Mbps bandwidth, while the latency of TCP Cubic is significantly increased at 12Mbps bandwidth. This is because the design of some existing congestion control protocols' objective functions is fixed for particular networks. Existing protocols, therefore, need to redesign the objective function when they are used in a new network environment.

B. The Challenges of Existing Congestion Control

As we have observed, most existing congestion control protocols are able to optimize performance for a particular aspect of the network and may thus achieve high performance in a particular network scenario but are unable to do this in other network scenarios. Moreover, there is also a convergence issue. To address these problems, we have designed a new congestion control protocol, MOCC, which is designed to tackle the following challenges.



(a) 12 mbps bandwidth

(b) 50 mbps bandwidth

Fig. 3: Performance over different bandwidth

Lack of multi-objective. Most learning-based congestion control protocols are performance-driven by means of pre-designed reward or objective functions, which are fixed. When new applications emerge, these protocols are unable to meet the performance requirements of these applications, resulting in a need to redesign objective functions and train new models. Consequently, it may be necessary to design a protocol over multiple competing objectives in order to meet the performance needs of different types of applications simultaneously.

Slow convergence. Setting a fixed init-cwnd for all network environments may reduce the network performance. For example, traditional congestion control protocols that use an additive-increase/multiplicative-decrease algorithm to increase the congestion window will result in lower throughput and under-utilization of network resources. We think that the factor causes convergence issues: lack of dynamic mechanisms to set the initialized congestion window value according to the network bandwidth.

Lack of generalization. Most congestion control protocols were designed for a specific network environment, given that different types of networks have different features (e.g., cellular networks experience high packet loss, but WiFi has high latency). Therefore, existing protocols can perform well in only one type of network. For example, Sprout [13] can lower the latency in cellular networks but not wireline connections. We accordingly require new congestion control protocols that can improve the performance in different network environments.

III. System design overview

MOCC is a multi-objective congestion control framework, an outline of which is presented in Fig 4. It consists of three major steps:

Interaction. In the Interaction, the agent observes the states of the network environment s_t and selects an action a_t, then observes a reward r_t. In the network environment, the sender sends data to the receiver and gets ACKs from the receiver according to the set congestion window size, which helps the MOCC agent determine the state of the network. To improve the convergence speed of MOCC, we design a novel dynamic initialization window method that sets the initialization congestion window depending on the bandwidth in question. For the agent, we design a dynamic terminal episode approach that applies multi-objective reinforcement learning to CC, based on the concept of a game of win-lose-tie. Through its use of the dynamic terminal episode mechanism, MOCC reflects the network data transmission more realistically and improves the model's training efficiency.

Multi-objective reinforcement learning. To meet the performance requirements of different application types, MOCC learns a policy simultaneously over multiple preferences by using multi-objective reinforcement learning. Each preference corresponds to the relative importance representation of multiple competing objectives.

Preference-oriented strategy. Depending on the relevant preference, we select the optimal model to meet the performance needs of different types of applications. We can set different preferences to realize a trade-off between the different performance objectives of throughput, latency, and packet loss.

IV. Design of MOCC

A. Agent

A Multi-Objective Markov Decision Process (MOMDP) can be represented as a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathbf{r}, \Omega, f_\Omega \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}\left(s' \mid s, a\right)$ is the transition distribution, $\mathbf{r}(s,a)$ is the vector reward function, Ω is the space of preference, and f_Ω is preference function, which take preference $\omega \in \Omega$ as inputs and outputs the scalar utility for each action in a given state, i.e., $f_\omega(\mathbf{r}(s,a)) = \omega^\top \mathbf{r}(s,a)$. Our goal is to maximize the utility and thus reveal the optimal policy for any preference $\omega \in \Omega$ set during inference.

State space. The state can be used to represent the network and should be carefully designed to reflect the congestion in an environment so that the agent can decide on the appropriate action according to what it has received. While a large number of states can more accurately reflect the state of the network, it can also lower the model's convergence speed. We, therefore, need to select the performance metric that best represents the network's condition as the state of MOCC. To this end, the state space of MOCC is represented as $\mathrm{S}_t = [\mathrm{Delay}(t), \mathrm{ACK\_rate}(t), \mathrm{Sending\_rate}(t), \mathrm{cwnd}(t)]$: here, $\mathrm{Delay}(t)$ is the delay in data transmission, and the change in delay can be used to reflect network congestion, while

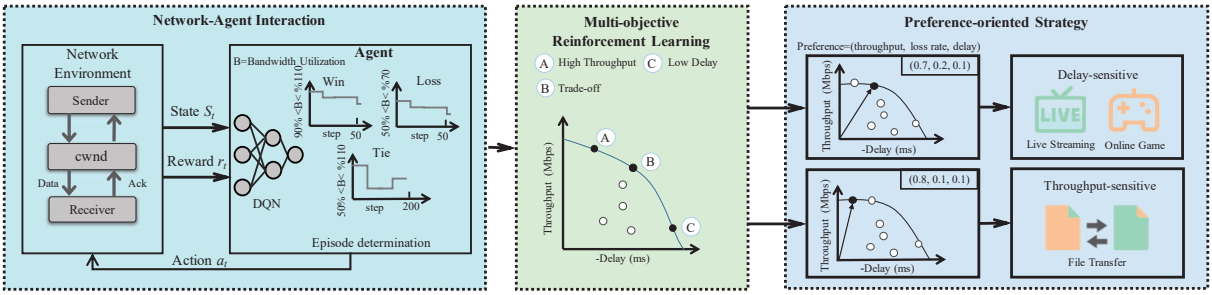


Fig. 4: The proposed multi-objective congestion control protocol is based on multi-objective reinforcement learning for the communication network MOCC. During the initialization phase, the sender sends data to the receiver according to the set initial congestion window size, improving the convergence speed of congestion control protocols.

ACK_rate(t) and Sending_rate(t) can represent the loss in the link; finally, cwnd(t) indicates the size of the congestion window (cwnd) calculated from the previous step.

Action space. The design of actions has a significant impact on the congestion control protocol performance, as the actions make it clear that the MOCC protocol should adjust the size of congestion window according to changes in the network environment. To improve the network performance, we select the action from $\{*0.5, -50, -10.0, +0.0, +10.0, *2.0, +50\}$ to adjust the congestion window. By selecting the appropriate action, we expect the congestion control protocol to increase the convergence rate during the initial data sending phase, quickly lower the data sending rate when network congestion occurs, and steadily adjust the size of congestion window based on changes in the network state during other phases.

Reward definition. The purpose of multi-objective reinforcement learning (MORL) is to learn policies on multiple objectives and relative importance (preferences). In comparison to traditional reinforcement learning, which optimizes for scalar rewards, the optimal strategy for multi-objective reinforcement learning relies on the relative preference of competing criteria. To test the robustness of the protocol, we perform a simple experiment to compare the average throughput of MOCC with other state-of-the-art protocols in different bandwidth environments. The reward function of MOCC network metrics is unprocessed. We can see that the throughput of the MOCC decreases to 0 in network environments above 60 Mbps. The throughput of other congestion control protocols increases with increasing bandwidth. This is because the bandwidth in the test network environment exceeds the bandwidth of the training environment resulting in a significant degradation in network performance. In addition, when the network latency is excessive, the reward function in the agent penalizes the latency too little to stop the congestion window from increasing in time which causes the imaginary timeout of sending data.

To overcome the aforementioned problem to successfully design a vectorized reward that comprises $[L(\text{Throughput}(t)), L(\text{Loss\_rate}(t)), L(\text{Delay}(t))]$ as three objectives, where $L(x) = (-10)^{-x} + 1$ is the activation function for each objective to increase penalties for network metrics that grow too fast and rewards slow changes in network metrics. For each objective, it has been normalized to match the significant interval of the activation function. As for the implementation

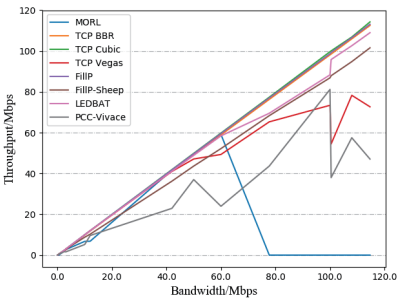


Fig. 5: Designed reward for agents with different bandwidth.

detail of normalization, we directly divide Throughput(t) by bandwidth, as well as divide Loss_rate(t) by timeout. Since Loss_rate(t) is already represented in zero to one, there is no need for normalization. By utilizing the aforementioned method, the model outperforms other state-of-the-art methods and shows its robustness in various network environments.

B. Initializing the Congestion Window Approach based on Different Bandwidths

As observed in the Section II, most congestion control protocols suffer from convergence issues due to the fact that they set a fixed value for the initial congestion window (init-cwnd). Moreover, the init-cwnd has a significant impact on the model convergence speed. One straightforward approach, which is used by most existing congestion control protocols, is to set the init-cwnd as a fixed value. However, these protocols fail to achieve fast convergence in different network scenarios due to their differing link capacities. Therefore, to improve the model's convergence speed, it is necessary to develop a dynamic adaptive initial congestion window approach to set initial congestion window values according to different network bandwidths. However, it is difficult for congestion control protocols to accurately obtain the bandwidth of the network.

To set the initial congestion window based on the different network bandwidths, we propose a novel dynamic adaptive initial congestion window method to set the value of init-cwnd with reference to the network bandwidth. In this method, we define the set of bandwidths $Bandwidth = (bw_1, bw_2, bw_3, ...bw_i..., bw_{n-1}, bw_n)$, where bw_i denotes the

bandwidth of the network. We determine $W_{init-cwnd}$ using the estimated bw_i, as in Eqn (1):

$$W_{init-cwnd} = b * bw_i, \tag{1}$$

where bw_i is the estimated bandwidth size, while b is the weight factor. Through experimentation, we determine that MOCC can achieve a faster convergence rate when b = 2.5. To estimate the size of bw_i, we determine the relationship between the average number of received acks Num_{ave} and the bandwidth bw_i, then estimate the size of bw_i according to the number of acks received in n steps. The number of acks received in n steps is Num_{ack}, and the average number of acks is $Num_{ave} = Num_{ack}/n$. The segmentation function F(Num_{ave}) is defined in Eq. (2) to determine bw_i:

$$F(Num_{ave}) = \sum_{j=1}^{n} Bandwidth_j I_{\{(a_{j-1},a_j)\}}\left(Num_{ave}\right), \tag{2}$$

where $I_{(a_{j-1},a_j)}$ is a characteristic function of the receive rate and is defined in Eq. (3), where (a_{j-1}, a_j) is the set of the receive rate.

$$I_{(a_{j-1},a_j)}(x) = \begin{cases} 1 & x \in (a_{j-1}, a_j) \\ 0 & x \notin (a_{j-1}, a_j) \end{cases} \tag{3}$$

C. Dynamic Terminal Episode Mechanism

To apply multi-objective reinforcement learning to congestion control, we take inspiration from game theory and design a terminal algorithm capable of ending an episode in the most appropriate way. An episode is a complete process in which the agent begins by performing a task in the network, then selects a series of actions in turn based on the network state and congestion control policy at each moment until it has finished sending the data. The length of the episode has a significant impact on whether the agent will learn the best model. Several current reinforcement learning-based congestion control protocols use a fixed number of episodes or a fixed time to determine the endpoints of the episode. However, this approach leads to a pseudo-terminal-state issue that introduces a considerable bias into the training. To address the pseudo-terminal-state problem caused by the application of multi-objective reinforcement learning algorithms to congestion control, we design a dynamic terminal episode method, inspired by the idea of winning, losing, and tying games, that makes decisions according to the changes in the network environment. The terminal method is able to end the episode at an appropriate time according to the changes in the bandwidth utilization, delay, and throughput of the network, thus improving the model's training efficiency.

To judge the agent's wins and losses, we define B as the bandwidth utilization. If 90% of bandwidth $\leq$ B $\leq$ 110% of bandwidth, while the delay is less than 0.7 times the timeout value, this indicates that the network resources are fully utilized, and we deem the network model to have achieved the goal of congestion control, thus increasing the number of wins by 1.

Algorithm 1: Termination Mechanism

Initialize the parameters of agent Win_{Num} = 0, the $Lose_{Num}$ = 0, M=50, L=50, and T=200;
if $90\%\ of\ Bandwidth \leq Throughput(t) \leq 110\%\ of\ Bandwidth\ and\ Delay(t) \leq 0.7 \times Timeout$ **then**
  Increase Win_{Num} by 1
else
  Win_{Num} = 0
if $50\%\ of\ Bandwidth \leq Throughput(t) \leq 70\%\ of\ Bandwidth$ *and* $0.7 \times Timeout \leq Delay(t)$ **then**
  Increase $Loss_{Num}$ by 1
else
  $Loss_{Num}$ = 0
if $Win_{Num} \geq M$ **then**
  Terminate the episode with a win;
else if $Loss_{Num} \geq L$ **then**
  Terminate the episode with a lose;
else if *($Win_{Num} \leq M$ and step=200) or ($Loss_{Num} \leq L$ and step=200)* **then**
  Terminate the episode with a tie;

When the number of consecutive wins exceeds M, we assume that the sender has finished sending the data and thus ends the episode. If 50% of bandwidth $<$ B $<$ 70% of bandwidth and the delay exceeds 0.7 times the timeout value, this indicates that the network has become very congested and that the network resources are not being fully utilized, thus incrementing the number of losses. When the number of consecutive losses exceeds L, we assume that the network is always congested and ends the episode. If the agent continues to execute T steps and the number of consecutive losses and wins both do not exceed L and M, we consider the policy to have finished sending the data and end the episode by recording a tie. For MOCC, we determined experimentally that larger and smaller M, L, and T both negatively influence the performance of the network; thus, as a satisfactory trade-off, M and L is set to 50 steps, while T is set to 200 steps. MOCC' dynamic terminal episode algorithm is summarized in Algorithm 1.

D. Learning Algorithm

Multi-Objective Reinforcement Learning Formulation. Compared to traditional Reinforcement Learning (RL) tasks, Multi-Objective Reinforcement Learning (MORL) tasks desire the agent to get policies that could simultaneously optimize more than one objective. When all objectives are relevant, we can combine all objectives together to form a single objective. When all the objectives are entirely unrelated, each of them can be optimized separately. However, if any of these objectives are conflicting, a given policy can achieve a trade-off between multiple conflicting objectives or maximize one of the key objectives. In MORL, the reward is a vector since each performance metric has its own related reward signal. In congestion control, congestion control protocols need to optimize several relevant performance metrics to improve the performance of different types of applications. Therefore, MORL can be applied to congestion control problems to meet

the performance requirements of different applications without the need to redesign the reward function.

MORL algorithms can be classified into multiple-policy, and single-policy MORL approaches [14]. For their part, the multiple-policy MORL algorithm is implemented by seeking a set of policies that closely approximate the Pareto frontier [15]. The goal of the single policy algorithm is to find an optimal policy that can satisfy multiple objectives preferences simultaneously among the multiple objectives specified by the user or defined by the application domain. However, learning a single policy that meets the performance needs of all applications is difficult across different network environments. We accordingly select a multi-policy MORL approach to solve the congestion control problem.

MORL can be considered as a combination of Multi-Objective Optimization (MOO) and Reinforcement Learning (RL) to solve sequential decision-making problems with multiple conflicting objectives [16]. The MOO problem of congestion control can be formulated as follows:

$$\begin{aligned} &\max M(O) = \left[m_1(O), m_2(O), \ldots, m_{m_f}(O)\right] \\ &\text{s.t. } g_i(O) \leq 0, i = 1, \ldots, a_g, \end{aligned} \tag{4}$$

where the "max" operator for a vector is a weighted scalar defined to maximize all network performance measures or the Pareto optimality. $O = [o_1, o_2, \ldots o_N]^T \in R^N$ is the variable vector to be optimised, while the functions $g_i(O)\,(i = 1, 2, \ldots, a_g)$ are the constraint function for the congestion control problem, and $m_i(O)\,(i = 1, 2, \ldots, a_f)$ are the objective functions.

In congestion control, we design a synthetic objective function $TQ(s, a, \omega)$ to address multiple network performance metrics by using the naïve approach, and the synthetic function represents the overall preference of the congestion control protocol. For each objective, it is updated with the following rules:

$$Q_i(s, a, \omega) = Q_i(s, a, \omega) + \left(r_i + \max_{a'} Q_i\left(s', a', \omega'\right)\right), \tag{5}$$

where $N \geq i \geq 1$. The single policy can be determined based on $TQ(s, a)$, which can be used for all objectives by considering a value space $\mathcal{Q} \subseteq (\Omega \to \mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}}$. The value space includes all bounded functions $Q(s, a, \omega)$-estimates of expected total rewards under m-dimensional preference (ω) vectors. In the congestion control task, we formulate the optimality operator as follows:

$$(TQ)(s, a, \omega) := r(s, a) + \gamma E_{s'\sim\mathcal{P}(\cdot|s,a)}(\mathcal{H}Q)\left(s', \omega\right), \tag{6}$$

where

$$(\mathcal{H}Q)(s, \omega) := \arg_Q \sup_{a\in\mathcal{A}, \omega'\in\Omega} \omega^\top Q\left(s, a, \omega'\right). \tag{7}$$

Let $Q^* \in \mathcal{Q}$ be the preferred optimal value function in the value space, such that:

$$Q^*(s, a, \boldsymbol{\omega}) = \arg_Q \sup_{\pi\in\Pi} \boldsymbol{\omega}^\top \mathbb{E}_{|s_0=s, a_0=a}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}\left(s_t, a_t\right)\right], \tag{8}$$

where the $\arg_Q$ takes the multi-objective value corresponding to the supremum. Then, $Q^* = TQ^*$.

Learning algorithm. As we aim to optimise a model capable of adapting to the entire space of Ω, we utilize a parameterised function to denote $\mathcal{Q} \subseteq (\Omega \to \mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}}$. We accomplish the goal by taking advantage of:

$$L^{\mathrm{S}}(\theta) = \mathbb{E}_{s,a,\omega}\left[\|y - Q(s, a, \omega; \theta)\|_2^2\right], \tag{9}$$

where

$$y = \mathbb{E}_{s'}\left[r + \gamma \arg_Q \max_{a,\omega'} \omega^\top Q\left(s', a, \omega'; \theta_k\right)\right], \tag{10}$$

which can be empirically estimated by sampling the transition (s, a, s', r) from the replay buffer.

Due to the large number of discrete solutions included in the optimal frontier in MOCC, that causes the landscape of the loss function to become non-smooth, so in practice, it is very challenging to go directly to optimizing L^{S}. To cope with this issue, we utilize the following auxiliary loss function L^{T}:

$$L^{\mathrm{T}}(\theta) = \mathbb{E}_{s,a,\omega}\left[\left|\omega^\top y - \omega^\top Q(s, a, \omega; \theta)\right|\right]. \tag{11}$$

By combining L^{S} and L^{T}, our final loss function is

$$(1 - \lambda) \cdot L^{\mathrm{A}}(\theta) + \lambda \cdot L^{\mathrm{B}}(\theta), \tag{12}$$

where ε is a weight to trade-off between L^{S} and L^{T}. More specifically, in order to transfer our loss function from L^{S} to L^{T}, we raise the value of ε from 0 to 1. This approach is very effective due to the fact that it uses the results of the previous optimization step as an initial guess at each update step. L^{S} first ensures that Q's forecast is near to any actual expected total reward, although it may not be optimal. For its part L^{T} provides an auxiliary pull along the direction, which has better effectiveness. We summarize the main algorithm of MOCC in Algorithm 2.

V. Evaluation

To evaluate the performances of MOCC, we conduct a series of experiments under different network scenarios by using Pantheon [8], Kernel (on top of Linux Kernel 4.13), and Mahimahi [17]. Pantheon can provide experimental results that can be reproduced and used to approximate real-world scenarios by using mahimahi shells. Moreover, Pantheon contains wrappers for many popular and research congestion control schemes, to which we can add our new congestion control schemes. It provides a popular and standard platform on which we can run these different congestion control protocols in a diverse testbed of the network on both wireless and wired networks using only a standard interface. After simple data processing, we can see the running result of these congestion control protocols. In these experiments, we compare MOCC and several other state-of-the-art congestion control protocols, namely TCP CUBIC [12], PCC Vivace [7], Fillp [8], Vegas [2], BBR [5], LEDBAT [3], and Fillp-sheep [8]. We also apply our algorithm in different network conditions, which helps us to understand and describe how MOCC performs.

Algorithm 2: Congestion Control based on MOCC Framework

Initialize replay buffer $\mathcal{D}_\tau$, network Q_θ, and $\varepsilon = 0$, which increases from 0 to 1
for *each iteration* **do**
  Reset the network environment;
  Initialize the $Sender_i$ and the $Receiver_i$;
  Successfully handshake between $Sender_i$ and $Receiver_i$;
  Sample a linear preference $\omega \sim \mathcal{D}_\omega$;
  for *each step* **do**
    if *iteration* = *2* **then**
      cwnd = 2.5 $\times$ $ACK\_rate(t)$;
    else
      Set termination of the episode by Algorithm 1.
      Sample an action ϵ-greedily:
      $a_t = \begin{cases} \text{random action in } \mathcal{A}, & \text{w.p. } \epsilon \\ \max_{a\in\mathcal{A}} \omega^\top Q(s_t, a, \omega; \theta), & \text{w.p } 1-\epsilon \end{cases}$;
      Apply action a_i to the congestion window cwnd;
      Receive a vectorized reward r_t and observe s_{t+1};
      Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}_\tau$;
      if *update* **then**
        Sample N_τ transitions $(s_j, a_j, r_j, s_{j+1}) \sim \mathcal{D}_\tau$;
        Sample N_ω preferences $W = \{\omega_i \sim \mathcal{D}_\omega\}$;
        Compute $y_{ij} = (\mathcal{T}Q)_{ij} =$
        $\left\{ r_j + \gamma \arg_Q \max_{\substack{a\in\mathcal{A},\\ \omega'\in W}} \omega_i^\top Q(s_{j+1}, a, \omega'; \theta), \right.$
        for all $1 \le i \le N_\omega$ and $1 \le j \le N_\tau$
        Update Q_θ by descending its stochastic gradient according to:
        $\nabla_\theta L(\theta) = (1-\varepsilon)\cdot\nabla_\theta L^{\mathrm{S}}(\theta) + \varepsilon\cdot\nabla_\theta L^{\mathrm{T}}(\theta)$
        Where $L^{\mathrm{S}}(\theta) = \mathbb{E}_{s,a,\omega}\left[\|y - Q(s,a,\omega;\theta)\|_2^2\right]$,
        $L^{\mathrm{T}}(\theta) = \mathbb{E}_{s,a,\omega}\left[\left|\omega^\top y - \omega^\top Q(s,a,\omega;\theta)\right|\right]$
        Increase ε

A. Choice of Parameters

To test the effect of different initialization windows on the model convergence speed, we set up a series of experiments related to the initial cwnd by comparing different forms of initializing cwnd, including setting the init-cwnd as a constant or using an initializing congestion window function that uses the mappings between the ACK rate and cwnd. The variable b is an essential parameter to improve the speed of convergence. Fig 6 presents the results of different forms of init-cwnd and different values of initializing congestion window function. We conduct this experiment in an environment with 12Mbps bandwidth and a delay of 20ms. First, we can conclude that the init-cwnd does affect MOCC convergence process. In this network scenario, the convergence cwnd is about 22. However, even if we set the cwnd to 20, it cannot achieve rapid convergence in this network scenario. Second, we find that the value of b does affect the convergence rate of MOCC. In Fig 6, MOCC can be seen to achieve convergence in a shorter time when b = 2.5 compared with other values of b. It also achieves better performance than results when init-cwnd is set a constant. Consequently, we can conclude that using a mapping function to set init-cwnd can utilize the network state to help MOCC achieve a faster convergence speed than initializing cwnd as a constant. By summarizing these results, we can determine that using a proper mapping to set the initial cwnd enables us to utilize the network information in order to help MOCC achieve faster convergence than is achievable when cwnd is initialized as a constant.

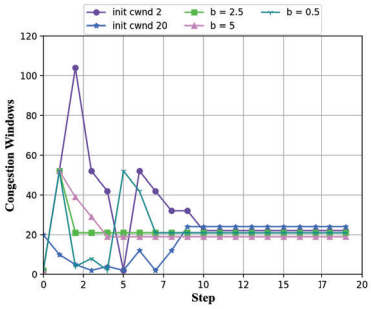


Fig. 6: Comparison of convergence steps between different methods of initializing the congestion windows. b is an essential parameter when using our mapping.

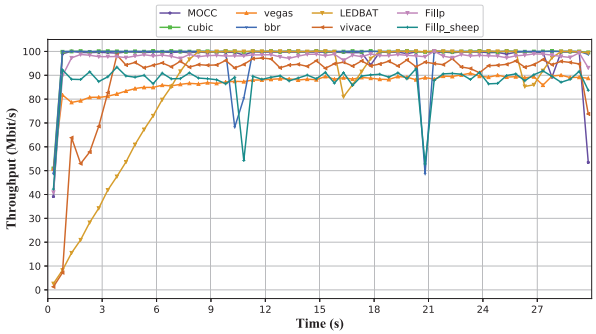


Fig. 7: Comparison of convergence between different protocols

To further demonstrate the advantages of our new approach to setting initial congestion windows, we conduct an experiment to compare the convergence rate of MOCC and other state-of-the-art protocols. We run different protocols in an environment of 100Mbps with a delay of 10 ms and a buffer size of 375KB. Fig 7 presents the results of the convergence process of different protocols. We focus on the beginning of the entire process to assess the convergence rate of each protocol. From Fig 7, we can see that in the first second of the whole process, MOCC has the highest throughput among all listed protocols. This means that MOCC exhibits a faster surge in throughput at the beginning of the process and that MOCC can achieve convergence in a shorter time compared with other protocols.

B. Performance in Stable Networks

To better describe the MOCC algorithm's performance and stability, we compare MOCC with other state-of-the-art congestion control protocols in two links provided by Mahimahi. One is a 12Mbps network with a delay of 10ms and a buffer size of 375KB; the other is a 50Mbps network with no delay and a buffer size of 375KB. The experimental results are presented in Fig 8.

From Fig 8, we can see that in the link of 12Mbps, MOCC has a throughput similar to that of Cubic, LEDBAT, Fillp, BBR, and Vegas, but achieves a lower delay. Moreover, MOCC still has a 14% higher throughput than Fillp-Sheep and 65% higher throughput than Vivace, while the delay of MOCC is lower

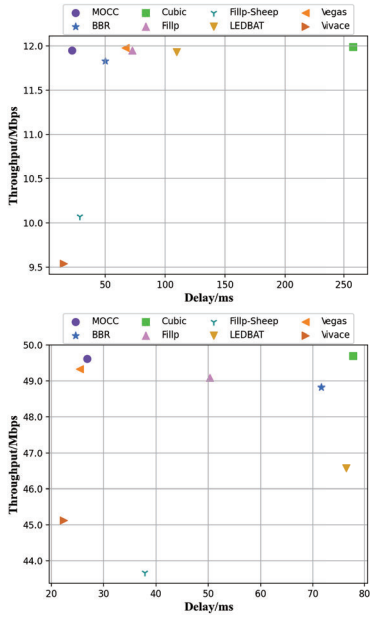


Fig. 8: Experimental results of different congestion control protocols running on stable networks of 12Mbps and 50Mbps

than all protocols except for Vivace. In the link of 50Mbps, MOCC has a similar throughput to BBR, Cubic, Fillp, and LEDBAT, but a lower delay than Cubic, Fillp, and LEDBAT. We can conclude that that the MOCC ranks in the top two of all tested protocols when running in a stable network scenario.

C. Changing Network in Real-life Situations

Excellent congestion protocols should provide outstanding performance for different types of applications in highly dynamic networks. A dynamic network presents a challenge to the flexibility and sensitivity of a congestion protocol. In fact, in our daily life, we most commonly use cellular networks when we use wireless devices. A cellular network is not as stable as a wired network, which means more changes and variations. We conduct a series of experiments to determine whether MOCC can provide the performance required for different types of applications when facing a dynamic network in daily life.

We select three typical cellular network scenarios: in a taxi, in a bus, and at home. The experiment is carried out using these cellular networks with a delay of 10ms and a buffer size of 375KB. Table I summarizes the experimental results. We can see that, under these three circumstances, MOCC can achieve high throughput by setting a high preference ratio and achieve a low delay by setting a low preference ratio. More specifically, in the taxi scenario, when we set the preference as $(0.975, 0.025)$, the throughput of MOCC is very close to that of Cubic and LEDBAT, which have the top two throughputs among all listed protocols. However, MOCC has a 44% lower delay than Cubic, 102% lower loss rate than Cubic, and a 20 % lower loss rate than LEDBAT. Moreover, when we set the preference to $(0.96, 0.04)$, MOCC has a lower delay and loss rate than all listed protocols except Vegas. In the bus scenario, by setting different preferences, MOCC can achieve the lowest loss rate or a lower delay compared with all other protocols except Vegas. In the home scenario, MOCC can also achieve the lowest loss rate and a very low delay. This demonstrates that MOCC can achieve an optimal trade-off between different performance metrics, thereby meeting the performance needs of different types of applications in the dynamic network scenarios by setting different preferences.

D. Performance at Different Buffer Sizes

The buffer is a vital resource for using by routers to store and forward packets and significantly impacts network performance: too large a buffer can easily lead to buffer flow, while a router with an insufficient buffer can drop packets and result in network resource underutilization. Therefore, we conduct extensive experiments to verify whether MOCC can efficiently utilize the network scenarios' bottleneck capacity with different buffer sizes through comparison with other state-of-the-art protocols.

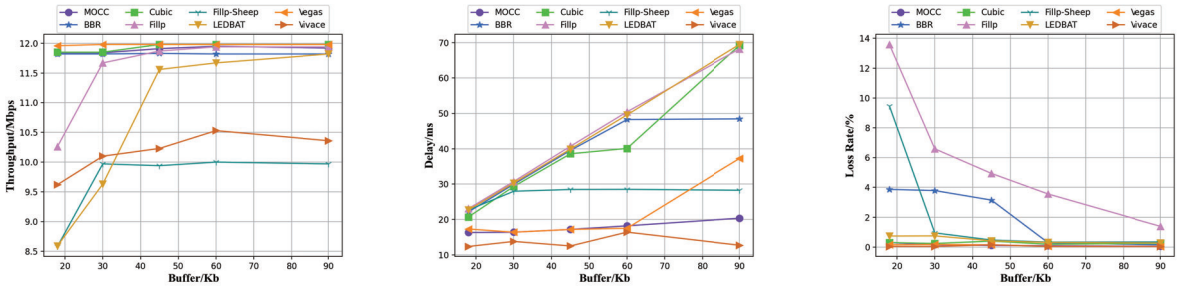
We conduct the experiment in an environment with 12Mbps bandwidth, a delay of 10ms, and buffer sizes varying from 18KB to 90KB. We train the MOCC model on a 12Mbps network with a delay of 10ms and a buffer size of 375KB. In Fig 9a, we can see that the throughput of some protocols like (Fillp, LEDBAT, Vivace, and Fillp-Sheep) drop quickly when the buffer size decreases; by contrast, MOCC can maintain a high throughput performance in networks with different buffer sizes. In Fig 9b, we can see that the delays of most protocols are affected by the increasing buffer size. However, MOCC can maintain a low delay when the buffer size increases, while other protocols (except Vivace) exhibit a higher delay when the buffer size increases. In Fig 9c, we can see that Fillp, Fillp-Sheep, and BBR have a high loss rate in the low buffer size environment, meaning that they are unsuitable for low buffer size networks. Compared to other protocols, MOCC has a 97% lower average loss rate than Cubic and a 255% lower average loss rate than LEDBAT. We can accordingly conclude that MOCC achieves outstanding performance and ranks among the best two protocols under different buffer conditions.

E. Performance in Unseen Network Conditions

As Pantheon is unable to implement network scenarios with varying bandwidths, we moved to the Linux Kernel to test the performance of MOCC under unseen network conditions. Now, we wish to evaluate the performance of MOCC over the unseen network environment, we choose a network with a minimum RTT of 20ms where bottleneck link bandwidth changes every 10s in the Linux kernel. Since the time scale of the change is 500 times the minimum delay, it is enough time for any protocol to converge to the equilibrium point. Fig 10 and Fig 11 show the changes in throughput and delay in the varying bandwidth, respectively. We can see that MOCC can achieve 99.8% link utilization, which is much higher than Vegas and BBR. As shown in Fig 11, the mean delay of MOCC is 26.02ms, which is 11.22% lower than BBR, which is 72.64% lower than Vegas, which is 91.35% lower than Cubic. This verifies that MOCC based on multi-objective reinforcement learning can achieve high performance over unseen conditions.

TABLE I: Comparison of throughput, delay and loss rate for different congestion control protocols in three different scenarios.

Congestion Control Protocol	Trace-taxi			Trace-bus			Trace-home		
	Throughput(Mbps)	Delay (ms)	Loss Rate(%)	Throughput(Mbps)	Delay (ms)	Loss Rate(%)	Throughput(Mbps)	Delay (ms)	Loss Rate(%)
TCP BBR	18.83	140.75	0.18	19.85	80.67	0.19	15.29	103.38	0.23
TCP Cubic	19.2	259.63	0.69	20.2	167.97	0.54	**15.52**	223.19	0.55
Fillp	18.77	160.98	1.3	19.84	147.57	5.43	15.33	194.44	1.31
Fillp-Sheep	16.96	84.39	0.57	17.74	56.88	4.9	14.47	81.34	0.77
LEDBAT	19.09	154.13	0.41	20.05	128.16	0.37	15.42	139.16	0.37
TCP Vegas	18.35	38.61	0.05	19.41	27.31	0.06	14.4	28.43	0.06
PCC-Vivace	7.84	106.58	2.03	10.68	184.86	10.85	8.53	126.66	1.02
MOCC(0.96/0.04)	15.48	**53.74**	**0.08**	12.68	**34.36**	0.06	11.3	**62.38**	**0.03**
MOCC(0.965/0.035)	16.84	105.3	**0.08**	12.96	40.58	**0.02**	12.25	67.05	0.12
MOCC(0.97/0.03)	17.62	108.95	0.25	15.52	83.07	0.04	13.09	90.16	0.19
MOCC(0.975/0.025)	**19.08**	180.58	0.34	**17.76**	103.67	0.36	**14.41**	166.96	0.17



(a) High throughput (b) Low delay (c) Low loss rate

Fig. 9: The performance of different protocols at different buffer sizes

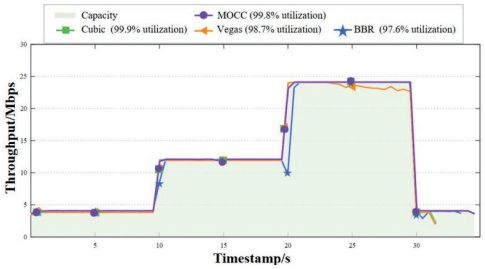


Fig. 10: MOCC achieves high throughput faces unseen network

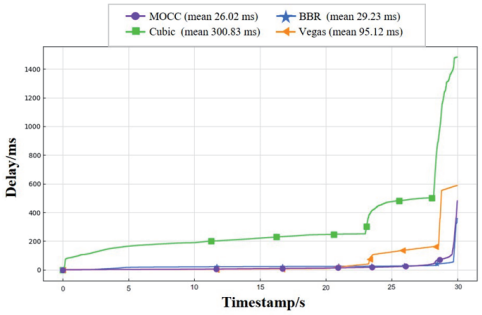


Fig. 11: MOCC achieves low delay faces unseen network

VI. Related Work

More than three decades of concerted research into congestion control has produced a large number of congestion control protocols, including rule-based protocols and learning-based protocols. Here, we aim to review several closely related works.

A. Rule-based Protocols

Many prior rule-based protocols employed several key congestion signals, including packet loss, and delay, as the key indicators of congestion, and then introduced a set of pre-defined rules to adjust the congestion window. Loss-based schemes such as TCP Taho [18], TCP Reno [19], TCP NewReno [20], and TCP Cubic [12] halve the congestion window when a given sender detects a packet loss. While loss-based schemes can achieve high throughputs, they do not satisfy latency-sensitive applications because they cannot guarantee lower transmission times. Moreover, they may mislead the protocols into making a decision, as it is difficult to determine whether the loss of a given packet is due to network congestion or a link error. Compared with loss-based schemes, delay-based schemes (such as TCP Vegas and Copa) determine the congestion window's size according to the increase of round-trip delays. Delay-based schemes control the congestion of links without being affected by link error but still find it challenging to accurately calculate accuracy.

B. Learning-based Protocols

Offline learning protocols. Offline learning protocols utilize off-line training and make similar assumptions regarding the network model. Winstein et al. proposed an offline optimization framework for congestion control, called Remy [21]. This approach attempts to iteratively build an offline mapping from all events to suitable actions by using defined assumptions about the network. Accordingly, Remy cannot achieve good performance when the assumptions it makes differ considerably from conditions in the real environment. Indigo [8], another offline optimization algorithm, employs an offline-trained neural network using a state-of-the-art imitation learning algorithm called DAgger [22] to perform training, labeling the correct action-state decisions via congestion-control oracle and LSTM

to determine the correct mappings between states and actions. An obvious limitation for both Indigo and Remy is that they cannot achieve good performance when their assumptions are significantly different from the real environment.

Online learning protocols. Unlike the offline approach, online learning protocols learn the relationship between rate control behavior and observed performance in an online manner. To avoid hardwired mapping between collected states and actions in traditional TCP variants, Dong et al. proposed PCC [6] and PCC Vivace [7], which select the best sending rate by employing online learning techniques; these techniques continuously attempt to modify sending rates on a small scale to approach better performance on the utility function. Despite PCC and PCC Vivace are capable of achieving good performance, online learning congestion control protocol requires an unacceptably long time to train an optimal policy.

RL-based protocols. RL-based protocols learn a congestion control policy by interacting with the environment; this policy can select appropriate actions to control the sending rate or congestion window depending on the state of the network. QTCP [23] combines Q-learning with TCP to design a novel learning protocol that can automatically learn effective strategies for adjusting the cwnd. Eagle [24] learns from an expert congestion control algorithm, BBR, and further uses reinforcement learning algorithms to train a generalized model by imitating expert algorithms' behavior rather than adopting a pure trial-and-error approach. Orca [25] uses reinforcement learning techniques combined with classical congestion control strategies to create a hybrid congestion control protocol. However, the design of the reward function for these methods is fixed, which results in a trained model that cannot be tuned to meet performance requirements.

VII. Conclusion

In this paper, we propose MOCC, a new congestion control protocol powered by multi-objective reinforcement learning to meet the performance requirements of different applications. When encountering different network scenarios, MOCC achieves fast convergence and high throughput by designing an initializing congestion window method to improve the convergence speed. Furthermore, we propose a dynamic terminal episode mechanism that draws on the game theory concepts of winning/losing/tying a game to end an episode in the appropriate way. Our experimental results reveal that, compared to a large number of recently proposed congestion control algorithms.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61972296, U20B2049, U20A20177, 61772377, 91746206), and the Fundamental Research Funds for the Central Universities (2042020kf0217).

References

[1] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proc. ACM SIGCOMM 2015 Conference*, vol. 45, no. 4. ACM, 2015, pp. 509–522.

[2] S. W. Brakmo, Lawrence S and L. Peterson, "TCP Vegas : new techniques for congestion detection and avoidance," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 24–35, 1994.

[3] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind *et al.*, "Low extra delay background transport (ledbat)," in *RFC 6817*, 2012.

[4] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 329–342.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, pp. 58–66, 2017.

[6] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: re-architecting congestion control for consistent high performance," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2015, pp. 395–408.

[7] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-learning congestion control," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 343–356.

[8] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for Internet congestion-control research," in *Proc. USENIX Annual Technical Conference*, 2018.

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

[10] L. Wu, S. Cao, Y. Chen, J. Cui, and Y. Chang, "An adaptive multiple spray-and-wait routing algorithm based on social circles in delay tolerant networks," *Computer Networks*, vol. 189, p. 107901, 2021.

[11] L. Wu, J. Yang, M. Zhou, Y. Chen, and Q. Wang, "LVID: A multimodal biometrics authentication system on smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1572–1585, 2019.

[12] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[13] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2013, pp. 459–471.

[14] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2014.

[15] M. Pirotta, S. Parisi, and M. Restelli, "Multi-objective reinforcement learning with continuous pareto frontier approximation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[16] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," *arXiv preprint arXiv:1908.08342*, 2019.

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

[18] J. Andren, M. Hilding, and D. Veitch, "Understanding end-to-end internet traffic dynamics," in *IEEE GLOBECOM*, vol. 2, 1998, pp. 1118–1122.

[19] M. Allman and V. Paxson, "E. blanton," tcp congestion control," RFC 5681, September, Tech. Rep., 2009.

[20] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The new reno modification to tcp's fast recovery algorithm," *RFC6582*, 2012.

[21] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 123–134, 2013.

[22] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," in *In AISTATS*, 2011.

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

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

[25] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual conference of the ACM SIGCOMM*, 2020, pp. 632–647.