Achieving Fairness Generalizability for Learning-based Congestion Control with Jury

Han Tian[1], Xudong Liao[2], Decang Sun[2], Chaoliang Zeng[3*], Yilun Jin[2], Junxue Zhang[2], Xinchen Wan[2], Zilong Wang[2], Yong Wang[2], Kai Chen[2]
[1]University of Science and Technology of China
[2]iSING Lab, Hong Kong University of Science and Technology [3]BitIntelligence

Abstract

Internet congestion control (CC) has long posed a challenging control problem in networking systems, with recent approaches increasingly incorporating deep reinforcement learning (DRL) to enhance adaptability and performance. Despite promising, DRL-based CC schemes often suffer from poor fairness, particularly when applied to network environments unseen during training. This paper introduces Jury, a novel DRL-based CC scheme designed to achieve fairness generalizability. At its heart, Jury decouples the fairness control from the principal DRL model with two design elements: i) By transforming network signals, it provides a universal view of network environments among competing flows, and ii) It adopts a post-processing phase to dynamically module the sending rate based on flow bandwidth occupancy estimation, ensuring large flows behave more conservatively and smaller flows more aggressively, thus achieving a fair and balanced bandwidth allocation. We have fully implemented Jury, and extensive evaluations demonstrate its robust convergence properties and high performance across a broad spectrum of both emulated and real-world network conditions.

CCS Concepts: • **Networks → Transport protocols**; • **Computing methodologies → Machine learning approaches**.

Keywords: Congestion Control, Reinforcement Learning, Transport Protocol

ACM Reference Format:
Han Tian, Xudong Liao, Decang Sun, Chaoliang Zeng, Yilun Jin, Junxue Zhang, Xinchen Wan, Zilong Wang, Yong Wang, Kai Chen. 2025. Achieving Fairness Generalizability for Learning-based Congestion Control with Jury. In *Twentieth European Conference on Computer Systems (EuroSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3689031.3696065

*Work done while Chaoliang was at iSING Lab @ HKUST.

EuroSys '25, March 30-April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1196-1/25/03
https://doi.org/10.1145/3689031.3696065

1 Introduction

Internet congestion control (CC) has become one of the most classic control problems in networking systems. Recently, inspired by great successes achieved by deep reinforcement learning (DRL) in various aspects (*e.g.,* games [29, 30, 38], computer systems [16], and networking [26, 27, 46]), CC researchers are exerting efforts on incorporating DRL into the CC problem [1, 18, 24, 41]. Due to the automatic learning process, DRL-based CC schemes provide superior performance over conventional protocols while freeing networking engineers from the tedious manual tuning of hard-wired control rules.

Despite being promising, DRL-based solutions are far from a silver bullet for the CC problem, primarily due to their inconsistency in meeting essential properties of CC tasks, particularly with regard to fairness. Internet congestion control tasks necessitate a guaranteed fair convergence property. However, the neural network models underlying DRL pipeline operate by stochastically updating and approximating the policy [22]. This stochastic nature complicates the integration of inherent properties such as fairness into the model.

To achieve fairness for learning-based CC schemes, previous protocols can be divided into three categories: (i) online exploration schemes that adopt trial-and-error online exploration (e.g., Vivace [8], Libra [9]), where the fairness is achieved through the convergence of a social concave game [12]; (ii) hybrid control schemes (e.g., Orca [1]) that introduce classic algorithms (such as Cubic) to attend to convergence properties; (iii) pure DRL-based schemes that add fairness-related metrics into the learning process to teach the model to behave fairly (e.g., Pareto [10], Astraea [23]).

However, current solutions fail to achieve efficient fairness across various network environments. On one hand, solutions in (i) require testing different sending rates/policies in real-time, observing the feedback, and deciding which action is good for the next iteration. The exploratory process consumes multiple RTTs, leading to slow convergence

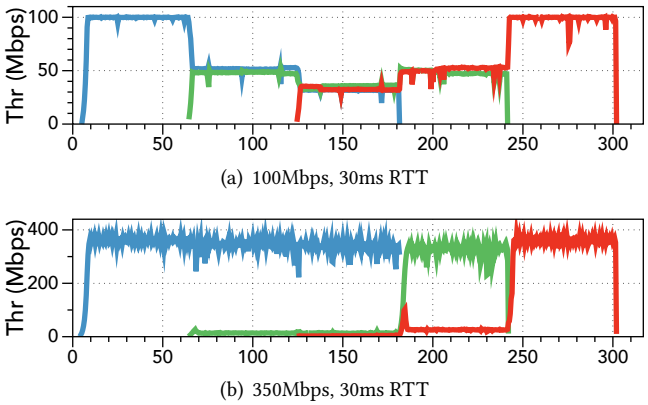
when the RTT is large. On the other hand, while trained to exhibit substantial fairness within controlled environments, learning-based solutions in (ii) and (iii) both show degraded convergence when applied to unseen network environments, thereby limiting their practical deployment on the open Internet. For example, Astraea adds a fairness metric into its learning reward to ensure fairness. When trained on links with bandwidth up to 100Mbps, it demonstrates perfect fairness in its training environment. However, the learned fairness behavior fails to generalize when applied to unseen environments with larger bandwidth (350 Mbps), as shown in Figure 1.

In this paper, we ponder a fundamental question: *is it possible to design a DRL-based CC scheme that achieves efficient fairness across various network environments, independent of its training domain?* We define the ability of a learning-based CC scheme to sustain fairness across unseen network environments as *fairness generalizability*, and focus on enhancing it in this paper.

To answer the question, we investigate the root cause of poor fairness generalizability observed in previous DRL-based schemes (§2.2). We identify a fundamental observation: there is an inherent trade-off between achieving fairness within a known network environment and extending this fairness to unexplored environments. To reach a fair equilibrium, the policy model requires bandwidth-related signals to learn differentiated flow behaviors, enabling large flows to yield bandwidth to smaller ones. However, these signals also introduce a divergence in the model input across network environments with varying bandwidth capacities, leading to generalizability issues. Previous works generally include bandwidth/throughput in the model input, therefore ensuring fairness in limited network scenarios at the cost of its generalizability.

Inspired by the dilemma, we argue that to achieve generalizable fairness, we need to decouple the task of ensuring fairness from the primary DRL model, therefore remaining unaffected by the learning process. We present Jury, a DRL-based CC scheme that achieves both theoretical and empirical generalizable fairness across various network environments. Jury is characterized by two key components.

- Jury designs an RL decision-making process that deliberately omits bandwidth-related signals. For the model input, we identify a set of observable network signals that are independent of the current flow bandwidth occupancy. For the model output, instead of directly generating the sending rate adjustment, it outputs a decision range. Trained with these invariant signals, flows competing with the same bottleneck receive uniform signal inputs and output the same decision range, regardless of their bandwidth utilization (§3.1).
- Jury further adopts a post-processing phase to dynamically decide sending rate adjustment from the decision range based on the flow's current bandwidth utilization estimation. We utilize sending rate changes and throughput responses to gauge the flow bandwidth utilization, which is then used to modulate the output sending rate. The phase differentiates the behavior of competing flows, making large flows more conservative and smaller flows more aggressive, thus promoting a fair and balanced allocation (§3.2).



(a) 100Mbps, 30ms RTT

(b) 350Mbps, 30ms RTT

Figure 1. Astraea cannot generalize its learned fairness in 3-flow case to unseen environments.

With robust assurance of fairness generalizability, Jury extends its efficient fairness from emulated training environments to real-world networks, adapting to diverse link characteristics and varying numbers of competing flows. Additionally, the sufficient network information encoded in the input signals ensures the learning capabilities of the DRL model (§3.1), thereby maintaining its consistently high performance across various network environments.

We have implemented and trained Jury in Linux servers supported by customized kernel modules (§4). Extensive evaluations (§5) show that Jury demonstrates consistently good fairness properties across i) various network environments with different link characteristics and ii) homogeneous and heterogeneous competing scenarios, including intra-protocol fairness, RTT fairness, and fairness between long-short flows (§5.1). Also, Jury maintains consistently high performance across various emulated environments and real-world Internet (§5.2) (bandwidth ranging from 5Mbps to 10Gbps, base one-way delay from 10ms to 400ms, and loss rate from 0 to 1.5%). The source code of Jury is available for the community at: https://github.com/tianhan4/jury.

2 Background and Motivation

2.1 DRL-based Congestion Control

Recently, deep reinforcement learning has gained broad interests in tackling congestion control challenges [1, 18, 24, 41]. DRL-based CC schemes are trained on network environments where the flow agent interacts with the environment

to gather training data. The flow agent works in a time-interval manner: for the t-th time interval, it observes network signals (*e.g.,* throughput, latency, and loss) from the environment as its input state $s_t \in \mathcal{S}$ and generates action $a_t \in \mathcal{A}$ to adjust the sending rate. The link state may change after the flow's action, and the flow agent will receive a new state s_{t+1}. During the interaction, the agent obtains a reward r_t from the environment and updates the policy based on r_t to build the mapping between the input state and the output action. The DRL agent refines its control policy during the training process to maximize the discounted cumulative expected reward $\mathcal{J} = \mathbb{E}(\sum_{t=0}^{T} \gamma^t r_t)$ during the lifetime of the flow, where γ is a discounted factor. Based on these data, a well-learned policy drives states to actions that lead to high performance.

Why reinforcement learning? DRL shows its superiority in congestion control with its data-driven learning nature, where the control policy is learned through real-world data without human intervention. This ability not only diminishes the reliance on the manual design of heuristic mappings and hyperparameter tuning but also enables continuous performance improvement through the adaptation to new data and experiences collected at end hosts [24, 47]. Contrastingly, traditional heuristic-based solutions are time-intensive and sophisticated processes that require substantial input and trial-and-error from experienced network professionals. The extensive efforts exerted by Google engineers in adapting the BBR protocol for diverse network environments like cellular links underscore this point [1].

2.2 Generalizable Fairness Issue

The generalizability of a DRL model defines its capability to perform well in environments that have never been seen during training. In the context of congestion control, it represents the ability to generalize to various network environments such as i) links with different bandwidths, delays, and loss rates; ii) congested links with various numbers of competing flows, and so on.

Several existing works have been focusing on improving the fairness of learning-based CC schemes. In this section, we highlight that they all fail to achieve generalizable fairness.

Online exploration schemes: Online exploration schemes utilize a stateless online learning paradigm to explore the optimal sending rate through trial and error [7–9, 28]. These schemes employ an exploration algorithm that dynamically adjusts the sending rate, observes the network feedback, and then selects the subsequent action. For instance, solutions like Allegro [7] and Vivace [8] incrementally increase or decrease the sending rate in small intervals, and then choose the direction that yields the best performance according to a predefined utility function. In contrast, Libra [9] compares the policies of a learning-based scheme and a classic scheme to determine the optimal sending rate. However, the exploratory process in these schemes generally requires several round-trip times (RTTs) to complete a single sending rate adjustment. This can lead to slow convergence on network paths with large RTTs, as shown in Figure 7(f). Furthermore, the slow response time of these online exploration schemes can hinder their ability to quickly adapt to dynamic bandwidth changes, as illustrated in Figure 12.

Hybrid control schemes: Another line of research attempts to combine learning-based algorithms with classic CC protocols, with the goal of leveraging the benefits of both approaches. For example, Orca [1] adopts a hybrid design that integrates both the Cubic and a DRL model to jointly control the sending rate. The underlying premise is that the hybrid scheme can benefit from the high performance of the DRL learning process as well as the good convergence properties of the Cubic protocol. However, the lack of a well-designed integration between the learning-based and classic components may lead to suboptimal performance in practical network scenarios, especially ones that are unseen during the training. The RL part may undermine the theoretical guarantees of fairness provided by Cubic, while the Cubic part can inadvertently affect the performance of the RL model by changing the sending rate according to its own internal state. As demonstrated in [23] and shown in Figure 7(h), Orca exhibits both unstable convergence and poor performance under lossy network conditions.

Fairness-oriented learning schemes: More recently, researchers have started to explore learning-based congestion control schemes that explicitly focus on incorporating fairness-related objectives into the learning process. The goal is to teach the control model to behave fairly towards competing flows directly. For example, Astraea [23] leverages multi-agent reinforcement learning techniques to incorporate the interaction between multiple flows as part of the reward signal during the learning process, incentivizing the agents to optimize for good convergence properties across competing flows. While these fairness-oriented learning schemes have demonstrated superior fairness within their trained operational regions, they often exhibit degraded convergence and performance when applied to unseen network environments, thereby limiting their practical deployment on the open Internet. For example, Astraea, when trained on 100 Mbps network links, has been observed to show poor fairness characteristics when applied to network conditions with significantly higher bandwidth capacity, as shown in Figure 1. Further online adaptation to new environments can hardly mitigate the problem, as the decentralized nature of Internet congestion control necessitates that DRL-based CC schemes are updated locally without global information of other competing flows, which hinders their ability to learn global fairness behavior in real-world networks. Another possible solution is to cover network conditions exhaustively during the training. However, as the real-world Internet exhibits various

complex behaviors due to various scheduling mechanisms, traffic shaping, and interactions with different congestion control protocols. The data/environment collection and training costs will become overwhelming. Furthermore, without explicit theoretical foundations, it is hard to diagnose and improve the policy generated by neural networks when they fail to achieve fairness.

To solve the problem, we dived into the learning pipeline of previous learning-based schemes. We found that their poor fairness generalizability in unseen network environments is primarily attributed to the difference of received network signals in trained environments and unseen ones. For example, when trained over 100 Mbps networks, the maximum observed throughput thr_{max} feature observed by Astraea may differ substantially for links with higher bandwidth capacities. As a result, the flow agent's behavior becomes unpredictable in these new network conditions, causing the well-learned fair policy to underperform or even fail.

Existing solutions have focused on normalizing the input network signals from various environments into a common feature space, in an attempt to reduce the input state divergence. Following this idea, various normalization techniques have been used in previous works [1, 18, 24, 41]. For instance, Aurora [18] uses normalized states including latency gradient and the ratio of latency to the minimum latency, and Orca [1] normalizes all throughput-related and delay-related features with the maximum observed throughput and the minimum observed one-way delay, respectively.

Following these techniques, we attempted to revise the original Astraea scheme by removing throughput-related features[1] from its input feature set. However, when we retrained the model with this modification, the model could hardly even learn to converge in training environments. After further investigation, we determined that the root cause was the lack of bandwidth-related input features. Attaining fairness requires the model to make distinct adjustments in sending rates among flows with different bandwidths. For example, under the same delay and loss signals, flows occupying larger bandwidth should behave more conservatively and back off their bandwidth to allow smaller flows to receive a fair share (this is also the core principle of AIMD). The analysis in the original Astraea paper [23] also shows that to achieve fairness, the Astraea flows are learned to respond to delays differently based on their current throughputs. The removal of the bandwidth-related features will impede the model's ability to differentiate the actions needed to reach a fair equilibrium.

Conclusion: We observe an inherent conflict between achieving fairness and ensuring its generalizability across diverse network environments. This conflict stems from the dilemma of whether to omit the bandwidth-related signals that are essential for the model to learn effective fair bandwidth allocation strategies.

[1]It includes the thr_{max} feature, the observed maximum throughput in the flow's history and the throughput ratio $\frac{thr}{thr_{max}}$.

2.3 Key Design Decisions

Inspired by the inherent conflict between fairness and generalizability, our key design decision is to decouple the task of ensuring fairness from the primary DRL model. We propose a novel DRL pipeline where the preservation of fairness properties is inherently embedded, addressing the limitations of prior approaches. Different from previous DRL-based CC schemes that blindly pour all available signals into the neural network model and use it as an end-to-end solution, we deliberately select input features for the DRL model to grant its fairness generalizability. Furthermore, we hand-crafted a preprocessing and postprocessing method for the model pipeline to ensure its fairness properties. Our design features two key components:

- We aim to design an RL decision-making process where *flows competing with the same bottleneck receive uniform signal inputs and generate the same output, regardless of their bandwidth utilization*. To achieve this, Jury deliberately removes bandwidth-related signals from the state input and instead utilizes a set of observable network signals that only relate to the bottleneck status. The process outputs a decision range, rather than a direct sending rate adjustment. By doing so, flows competing with the same bottleneck always receive uniform signal inputs and output the same decision range, regardless of their individual bandwidth utilization. This not only preserves fairness generalizability, but also constructs a consensus point between competing flows as the basis for the subsequent fairness-oriented component (§3.1).
- Jury adopts a post-processing phase to modulate the sending rate changes between competing flows, ensuring fairness. Specifically, based on the decision range generated by the DRL model, this phase dynamically adjusts the sending rate for each flow based on its current bandwidth utilization. We first estimate the flow's bandwidth utilization (throughput/link capacity) using its historical sending rate changes and throughput responses. Then, this bandwidth utilization estimate is used to modulate the output sending rate, such that larger flows get lower sending rates and smaller flows get higher sending rates. In this way, the phase differentiates the behavior of competing flows, making large flows more conservative and smaller flows more aggressive, thus promoting a fair and balanced allocation (§3.2).

As a result, we can attain a balanced and fair distribution among flows even with normalized input features, achieving both fairness and its generalizability across network environments. The pre-processing and post-processing parts enable the DRL-based model to achieve fairness without relying on explicit bandwidth utilization features, effectively incorporating "classic CC wisdom" into the ML-based solution,

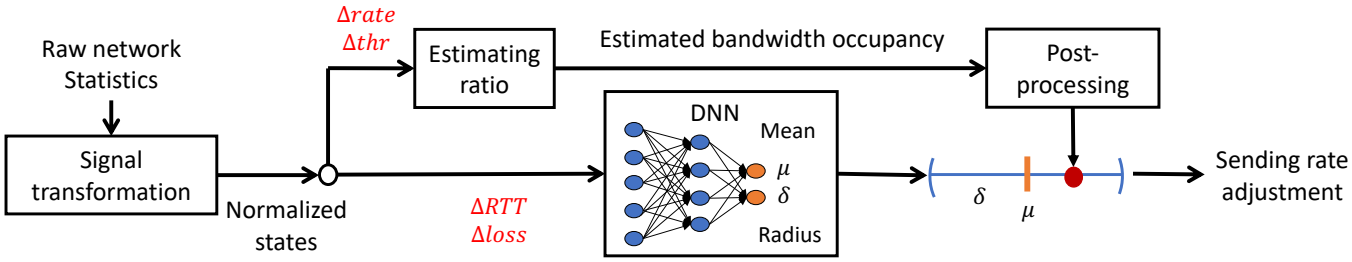


Figure 2. The DRL pipeline of Jury. Preprocessing feature transformation ensures the generalizability of the pipeline, and the post-processing function ensures the fairness property.

allowing the DRL model to focus on optimizing the performance goal (e.g., high bandwidth and low delay). Given sufficient informed signals, the pipeline continues to exhibit strong performance across both training and unseen testing network conditions, as guaranteed by the normalization technique and shown in the evaluation sections (§5).

3 Design

Based on the observation and design decisions, we design our fairness-generalizable DRL-based CC algorithm, Jury. Fig. 2 overviews the pipeline of Jury. It consists of three blocks: i) the signal transformation function; ii) a DRL model, and iii) a post-processing function. During each time interval, a flow acquires raw network statistics s_{raw} from the environment, including throughput, latency, loss, and so on. These raw signals are fed into Jury's signal transformation function to generate normalized observable states, which are then bifurcated into two separate components.

For the DRL model's input, we select latency and loss network signals, as they describe the bottleneck congestion level and are agnostic to flow bandwidth occupancy. This ensures that the DNN models for all competing flows receive identical input states and yield the same output actions, characterized by a decision range with mean μ and radius r. Also, all the signals are normalized to ensure generalizable performance across various network environments.

On the other path, bandwidth-related states (including sending rate change and the corresponding throughput change) are employed to gauge the flow's bandwidth occupancy ratio. This ratio informs the post-processing phase in determining a specific point within the (μ, r) range, which is then converted into an action for multiplicative rate adjustment.

The pipeline works as if plugging in an adaptor into the DRL model, normalizing the state input and then 'denormalizing' the output to rescale the sending rate adjustment according to the flow's bandwidth usage. This allows distinct rate adjustments for different bandwidth flows, leading to a fair equilibrium point.

We describe the details of each component in the pipeline of Jury in the following sections, including its signal transformation (§3.1), action generation pipeline (§3.2), reward definition (§3.3), signal processing mechanism (§3.4) and the RL training algorithm (§3.5).

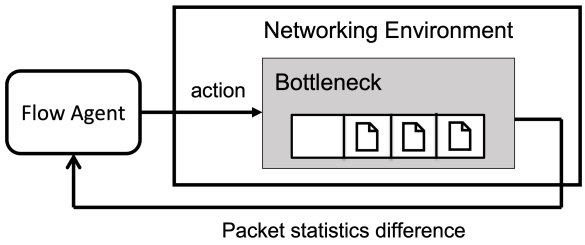


Figure 3. The action-feedback signals of a Jury flow.

3.1 Signals Transformation

It is non-trivial to locate normalized signals that encode sufficient information for control decision-making. In this section, we design Jury's signal transformation block that both provides rich network information for the optimal sending rate control and preserve the fairness generalizability of policies learned with them.

As shown in Fig. 3, Jury's signals are based on an action-feedback mechanism that actively detects the current flow and queue status in the bottleneck. Specifically, Jury records the enforced sending rate adjustments and the corresponding packet statistics changes. For an action a_{t-1} enforced at the t-th time interval that updates the sending rate from x_{t-1} to x_t, Jury tracks the outbound packets during this interval and collects the following metrics based on their ACKs:

- The ratio of throughput change to the throughput in the last time interval $\frac{thr_t - thr_{t-1}}{thr_{t-1}}$, where thr_t is the average throughput for packets sent at the t-th time interval.
- The difference in RTT of two adjacent time intervals $RTT_t - RTT_{t-1}$, where RTT_t is the average RTT for packets sent at the t-th time interval.
- The ratio of (1-loss_rate) between the two time intervals $\frac{1-L_t}{1-L_{t-1}}$, where L_t is the average loss rate for packets sent at the t-th time interval.
- The multiplicative sending rate change $a_{t-1} = \frac{x_t}{x_{t-1}}$.

We normalize throughput and loss with their previous values, only using their multiplicative changes. For the RTT signal, we use the reductive change, as the change in RTT

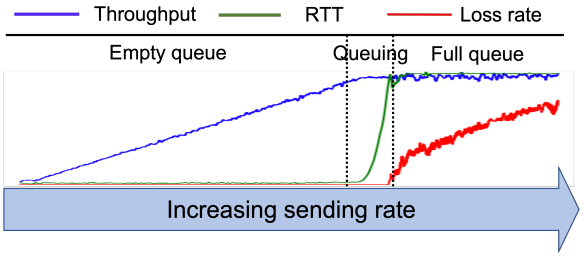


Figure 4. The packet statistics that change with increasing sending rate depend on the queue status (all statistics are scaled to [0,1] for visualization.).

inherently represents the ratio between the overall sending rate and the link capacity:

$$\Delta RTT = RTT_t - RTT_{t-1} = \Delta t \cdot \frac{\Sigma_i x_{i,t} - c}{c}, \tag{1}$$

where $\Sigma_i x_{i,t}$ is the overall sending rate of all flows at t-th time interval, Δt is the time interval range, and c is the link capacity. Given a constant time interval, ΔRTT represents the imbalance between the overall sending rate and the available link capacity. This normalization approach ensures that the collected metrics are insensitive to the specific network environment. Models fed with these signals will focus on learning the relative changes in the throughput, RTT, and loss rates instead of their absolute values in any specific network environment, allowing the learned policy to generalize well across different network conditions.

Besides their generalizability, the above action-feedback based signals probe sufficient information about the current network environment to enable intelligent control. Here we show how a flow can i) learn to optimize its throughput while reducing latency through signals ($RTT_t - RTT_{t-1}$, $\frac{1-L_t}{1-L_{t-1}}$), and ii) estimate its bandwidth occupancy in multi-flow scenarios using ($\frac{x_t}{x_{t-1}}$, $\frac{thr_t - thr_{t-1}}{thr_{t-1}}$).

For the latency and loss difference signals $RTT_t - RTT_{t-1}$ and $\frac{1-L_t}{1-L_{t-1}}$, Fig. 4 shows the growths of throughput, latency, and loss rate with a single flow on a link (100Mbps, 30ms RTT, 750KB buffer size) that keeps increasing its sending rate. We note that different metrics exhibit heterogeneous behaviors across different stages of queue building: i) when the link is under-utilized and the queue is empty, the throughput increases with the sending rate while its RTT and loss rate remain unchanged; ii) when packets start queuing in the bottleneck, increasing the sending rate will not affect the throughput anymore, and RTT starts to increase; iii) after the queue is full, continuing to increase the sending rate with not affect the RTT anymore, and packet loss starts to increase. Therefore, through recognizing different responsive behaviors of throughput, RTT and loss rate concerning sending rate modifications, the flow agent senses the current queue-building phase and is thus able to tune its sending rate towards the optimal operating point. This optimal point is the junction of the "empty queue" and "queuing" phases, where the link is fully utilized with almost no queue.

We then elaborate how sending rate and throughput difference signals $\frac{x_t}{x_{t-1}}$ and $\frac{thr_t - thr_{t-1}}{thr_{t-1}}$ help estimate flow's bottleneck bandwidth occupancy. Fig. 5 depicts how the flow's throughput changes at different bandwidth occupations, when its sending rate is increased by 10% in a 2-flow link (100Mbps, 30ms RTT, 750KB buffer size). We observe that when the flow occupies less share of the link capacity, increasing the sending rate leads to a larger increase in throughput. Based on this characteristic, we can estimate the bandwidth occupation ratio of flows based on the relations between its sending rate changes and throughput changes through the following congestion control modeling.

When multiple flows fully occupy the bandwidth of a bottleneck, for a specific flow, the feedback of throughput change corresponding to its sending rate change is related to its existing share of the link bandwidth. For a specific flow, its share of link bandwidth is related to the ratio of its sending rate to the overall sending rate of all competing flows:

$$thr_{i,t} = \frac{x_{i,t}}{\Sigma_{j \neq i} x_{j,t} + x_{i,t}} \cdot c. \tag{2}$$

When increasing its sending rate via action $x_{i,t+1} = a_{i,t} \cdot x_{i,t}$, the updated throughput should be

$$thr_{i,t+1} = \frac{a_{i,t} \cdot x_{i,t}}{\Sigma_{j \neq i} x_{j,t} + a_{i,t} \cdot x_{i,t}} \cdot c. \tag{3}$$

Therefore,

$$\frac{thr_{i,t+1}}{thr_{i,t}} = \frac{a_{i,t}(\Sigma_{j \neq i} x_{j,t} + x_{i,t})}{\Sigma_{j \neq i} x_{j,t} + a_{i,t} x_{i,t}} = \frac{a_{i,t}}{1 + (a_{i,t} - 1)\frac{x_{i,t}}{\Sigma_j x_{j,t}}} = \frac{a_{i,t}}{1 + (a_{i,t} - 1)\frac{thr_t}{c}} \tag{4}$$

We can further get the flow bandwidth occupancy ratio from the sending rate change and its corresponding throughput change:

$$ratio_{bw} = \frac{thr_t}{c} = \frac{a_{i,t} - \frac{thr_{t+1}}{thr_t}}{\frac{thr_{t+1}}{thr_t} \cdot (a_{i,t} - 1)} \tag{5}$$

Here, thr_t denotes the flow throughput at t-th time interval, c denotes the link capacity of the bottleneck. This estimation can be directly calculated from our action-feedback signals[2]. With these bandwidth occupancy estimates, Jury can recalibrate the aggressiveness of sending rate modifications among different competing flows to achieve a fair equilibrium point.

3.2 Rate Adjustment Generation

Following the signal transformation process, the two parts of the transformed signals are bifurcated into the DRL model and its post-processing function, respectively. The latency

[2]BBR has also given similar analysis for its fairness property [40].

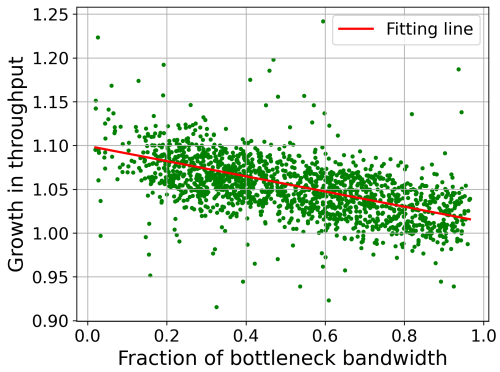


Figure 5. For a flow occupying different portions of the link capacity, we observe its throughput change when increasing its sending rate by 10%.

and loss differences ($RTT_t - RTT_{t-1}$ and $\frac{1-L_t}{1-L_{t-1}}$) are used by the neural network model. By relying only on RTT and loss signals, the model ensures that all competing flows at the bottleneck receive identical input states. This input uniformity leads to consistent outputs from DRL models for all flows competing at the same bottleneck. We deliberately exclude the throughput and sending rate difference signals from the input so that the model cannot infer the flow's throughput or bandwidth occupancy, preventing the learning policies from violating the fairness guarantee established by the post-processing phase. The output from the neural network model comprises a pair of values: the mean μ and the radius δ, which delineate an action range.

Concurrently, the throughput and sending rate differences ($\frac{x_t}{x_{t-1}}$ and $\frac{thr_t - thr_{t-1}}{thr_{t-1}}$) are used to estimate the flow bandwidth occupancy based on Equation 5, which is then used in the post-processing phase to determine the precise rate adjustments for different flows within the given action range (μ_t, δ_t) as following:

$$a_t = \mu_t + (1 - 2 \cdot ratio_{bw}) \cdot \delta_t \tag{6}$$

This approach ensures that flows with a higher bandwidth occupancy always produce smaller actions, leading to a gradual relinquishment of bandwidth to lesser-occupying flows. This process continues until a fair equilibrium in bandwidth sharing is achieved, establishing the convergence guarantee of Jury. Furthermore, the insensitivity to link characteristics of Jury's signals enables the fairness to generalize to flows with heterogeneous base delays and random loss rates due to their different network paths, as shown in §5.1.2.

Then, similar to previous DRL-based CC schemes [1, 18, 24, 41], Jury adjusts the congestion window and then calculates a proportional pacing rate based on the output action. It adopts a multiplicative sending rate adjustment. Specifically, the action a_t generated by the DRL model in Jury in each time interval updates the congestion window as follows:

$$cwnd_{t+1} = \begin{cases} cwnd_t \cdot (1 + \alpha \cdot a_t) & a_t \geq 0 \\ cwnd_t / (1 - \alpha \cdot a_t) & a_t < 0 \end{cases}, \tag{7}$$

where α is a hyperparameter that controls the policy aggressiveness. Then, we calculate the pacing rate based on the updated congestion window and the average RTT observed in the last interval:

$$x_{t+1} = \frac{cwnd_{t+1}}{RTT_t}. \tag{8}$$

3.3 Reward

The reward function is pivotal in defining the learning objective for the flow agent's control policy. Jury's reward function employs local signals to enable further policy adaptation. Inspired by Vivace's utility framework [8], the reward function for Jury is formulated as follows:

$$R = (ratio_{bw})^{\zeta} - ratio_{bw} \cdot (\beta_1 \cdot (RTT - RTT_{min}) - \beta_2 \cdot \frac{1 - L_t}{1 - L_{min}}), \tag{9}$$

where $0 < \zeta < 1$. The reward definition focuses on increasing the flow bandwidth occupancy while decreasing incurred latency and packet loss. The terms RTT_{min} and L_{min} denote the observed minimum RTT and loss rate, respectively. It is plain that both $(RTT - RTT_{min})$ and $\frac{1-L_t}{1-L_{min}}$ can be derived from the flows's signal history. The coefficients β_1 and β_2 represent the weight of latency and packet loss. We have tuned the weights (Table 2) for the learned policy to achieve a good balance among various criteria (throughput, delay, and loss), as shown in the evaluation section. For a specific CC objective with new preferences, Jury would need to be retrained. MOCC [24] provides a multi-objective DRL-based CC framework that can adapt to different preferences simultaneously without retraining, which can be adopted in our method to fit networking applications with various requirements.

Echoing the action generation process, the reward structure is also designed to incentivize smaller flows to increase their throughput more aggressively than larger flows. The reason is twofold: first, the concave throughput term $(ratio_{bw})^{\zeta}$ ensures higher rewards for small flows when increasing their throughput compared to larger flows. Second, as the throughput ratio also influences the penalty terms, smaller flows incur lesser penalties due to latency inflation and packet loss, recognizing that they have a limited impact on queue length and thus bear less responsibility for congestion. According to Vivace [8], such a concave utility function ensures the convergence of its online learning process to a fair equilibrium. While Jury's fairness property is primarily upheld by the post-processing phase, we find that the usage of the reward function in Equation 9 practically stabilizes Jury's learning process.

3.4 Signal/Action Processing

To improve the robustness of Jury, we incorporate several preprocessing/post-processing techniques on top of the DRL scheme to keep empirical signals and responses aligned with our generalization component.

Signal Averaging and Filtering As illustrated in Figure 5, even with consistent sending rate alterations, network signal changes vary due to network jitters. To mitigate the issue, Jury employs a moving average filter to smooth the estimation. Furthermore, we establish upper and lower bounds for these signals to filter outliers. Samples that fall outside these boundaries are adjusted to the corresponding threshold values. We also observe that the noise will not severely affect the DRL-CC pipeline performance. The reason may be due to the incremental process of the sending rate adjustment in congestion control, where the output sending rate changes modulated by the noisy estimation will also be averaged over time.

Exploration Action We adopt specific rules for output actions to meet innate exploration requirements from action-feedback signals. We encourage the agent to increase/decrease the sending rate to probe the current network condition. Specifically, if the output action a_t is near 0 ($e_{lower} < a_t < e_{upper}$), we assign a high likelihood to modify the action to either 1 or -1 (with the same probability, so the expectation remains unchanged). This adjustment compels the flow to either maximally boost or reduce the sending rate so it can investigate more obvious responses tied to deliberate fluctuations in sending rates, enabling more informed policy decision-making for the RL model.

Enhance Statistics Significance To collect consistent and robust statistical signals, Jury just keeps maximally increasing the sending rate when packets inside one interval are less than the pre-defined threshold, ensuring sufficient samples to generate statistically significant signals and thus, inform a reliable decision-making process. Additionally, this mechanism can also i) work as a slow-start phase and ii) allow short flows to circumvent the DRL inference overhead.

3.5 Training

For the training algorithm, Jury adopts Deep Deterministic Policy Gradient (DDPG), a prevalent model-free, off-policy DRL algorithm used in previous DRL-based CC schemes [1, 24, 41]. Jury utilizes the actor-critic framework to train the agent's policy, which involves both a critic model that estimates the expected cumulative rewards (the action-value function $Q^{\pi_\theta}(s,a) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t \mid a, s]$) and an actor model that decides the actions.

During training, the actor interacts with the network environment, gathering trajectories composed of tuples (s_t, a, r, s_{t+1}). After each run of trace, we perform an update operation. For every update step, we sample tuples from the collected training data in batches and update the actor and critic models. Jury updates the actor's policy π_θ by minimizing the following objective function:

$$\mathcal{J}(\theta) = \mathbb{E}\left[Q_\omega(s, \pi_\theta(s))\right], \tag{10}$$

where $Q_\omega(s,a)$ is the output of the critic that estimates the action-value function $Q^{\pi_\theta}(s,a)$ under the current policy. This estimation guides the actor in selecting actions that lead to higher rewards. On the other hand, the critic is trained to improve its value estimation by estimating the difference between subsequent states. It minimizes the following objective function:

$$\mathcal{L}(\omega) = \mathbb{E}_{s,a,r,s'}\left[\left(Q_\omega(s,a) - r + \gamma Q_\omega(s', a')|_{a'=\boldsymbol{\pi_\theta}(s')}\right)^2\right]. \tag{11}$$

After calculating the gradients, Jury updates the actor and critic models with learning rates σ and η, respectively:

$$\theta \leftarrow \theta + \sigma \nabla_\theta \mathcal{J}(\theta), \qquad \omega \leftarrow \omega - \eta \nabla_\omega \mathcal{L}(\omega). \tag{12}$$

Similar to previous methods, we stack signals from a window of intervals as the input state of the model to enrich the input information. We also adopt several RL-related training techniques used in Twin Delayed Deep Deterministic Policy Gradient (TD3) [14], including clipped double Q-learning, delayed policy updates, and target policy smoothing regularization. These techniques help reduce the variance of the critic model's value estimation.

Bandwidth	Base RTT	Buffer size	Loss rate
20-100 Mbps	10-60ms	0.8-1.5 BDP	0-0.1%

Table 1. DRL-based methods training environment.

4 Implementation

We implement a fully functional Jury prototype in Linux. It consists of the CC kernel module in the Linux TCP network stack and the RL agent in the userspace. The kernel CC module collects the network signals and sends them to the RL agent in the userspace, which then returns the control action to the kernel. The kernel module uses the action to change the congestion window and pacing rate. The cross-space communication channel is implemented with netlink [37]. For the DRL model design and training, we use the RL framework DI-engine [11] based on PyTorch [34] that provides support for various deep reinforcement learning algorithms as well as the capability to design custom policies. For the inference part, we implement the inference service entirely in C++ using the high-performance PyTorch C++ API. This approach is more lightweight than previous learning-based implementations, such as Orca [1], and avoids the need for cross-process communication between a C++ client and a Python-based

Name	Value
control time interval	30 ms
actor learning rate (σ)	5e-4
critic learning rate (η)	1e-3
discount factor (γ)	0.98
batch size	64
model update interval (second)	5
action control coefficient (α)	0.025
RTT scale coefficient (β_1)	1e-5
loss scale coefficient (β_2)	5

Table 2. Training hyperparameters in Jury.

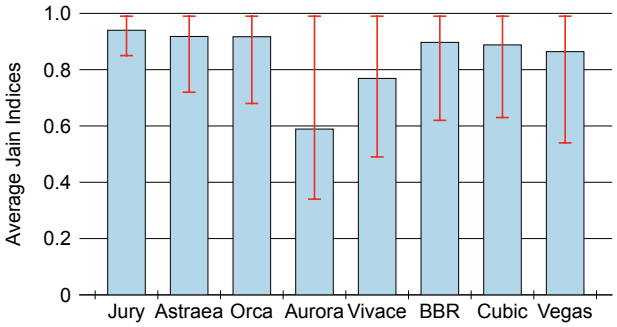


Figure 6. The average Jain Index of homogeneous competing flows with 5th and 95th percentiles.

inference service. For the neural network models, we adopt a model structure with two 128-dimensional fully connected layers. We train Jury on emulated network environments implemented with Mahimahi [32] and Pantheon-tunnel [45] supporting customized link capacity, base delay, and random loss.

We use a distributed training setup with 8 parallel processes (actors) to collect the training experience, and a single process (critic) to update the model with the collected data every 5 seconds. This allows us to efficiently gather diverse training data while keeping the model update process centralized.

We train and evaluate Jury on a Linux server with 80 CPU cores, 256 GB of RAM, and an NVIDIA GeForce RTX 3090 GPU. We leverage the GPU for neural network model updates during the training phase, but use only the CPU for inference, making the policy deployment more feasible on common end-host devices. The training process converges within 4 hours. We have carefully chosen and tuned all the training hyperparameters to stabilize the learning process of Jury, and we list their values in Table 2.

5 Evaluation

In this section, we evaluate Jury with emulations and real testbed experiments to validate its fairness generalizability and consistent high performance across various network conditions. We demonstrate Jury's convergence generalizability in §5.1 and performance in §5.2. We compare Jury with several baselines, including recently proposed learning-based CC schemes (Astraea [23], Orca [1], Aurora [18], Vivace [8]) and classical heuristics-based CC schemes (Cubic [15], BBR [4], Vegas [3]).

To avoid the influence of the range of the training environment, we retrain all learning-based schemes with the same training environment region as shown in Table 1. Throughout the training phase of these DRL-based schemes, we also simulate a varying number of homogeneous and Cubic concurrent flows (ranging from 2 to 10) into the environment to facilitate the learning of robust convergence properties.

5.1 Generalizable Fairness

In this section, we evaluate Jury's fairness generalizability in terms of intra-protocol fairness, RTT-fairness, fairness under multi-bottleneck scenario, and friendliness.

5.1.1 Fairness among Jury Flows. To evaluate the generalizability of Jury's fairness in different conditions, we set up experiments with three homogeneous flows, each running for 180 seconds and starting 60 seconds apart. We conduct 60 repetitions of this experiment on emulated links, representing a range of network conditions with bandwidths from 20Mbps to 400Mbps, base one-way delays from 10ms to 75ms, and loss rates up to 0.3%. The Jain's Fairness Index, along with the 5th and 95th percentiles, is presented in Fig. 6. The optimal Jain's index is 1, which indicates perfect fairness where all flows have equal bandwidth immediately and stay constant. We observe that Jury achieves the highest average Jain Index (0.94) among the baselines with also the highest 5th percentile (0.82). This performance underscores Jury's robust fairness across diverse network environments, confirming that our design achieves generalizable fairness across various network environments.

Fig. 7 further demonstrates the throughput dynamics of competing flows under various network conditions. As observed in Fig. 7(a), 7(b), 7(c), and 7(d), Jury displays consistent fairness behaviors across different link capacities, delays, and loss rates. Its convergence speed is a little slower in large BDP links due to limited rate adjustments per interval and delayed feedback in high RTT scenarios. In contrast, existing learning-based schemes show poor fairness in specific conditions. Astraea, for instance, struggles with fairness outside its training region (Fig. 7(e)), as it lacks of a guaranteed fairness learning process. Vivace and BBR exhibit slow convergence in scenarios with large RTTs and packet loss (Fig. 7(f), 7(g)), as it requires several RTTs to adapt to network changes. Orca, despite demonstrating good fairness in various conditions (average Jain Index=0.91), faces challenges on lossy links where its underlying Cubic scheme also underperforms, leading to both reduced link utilization and poor fairness (Fig. 7(h)). The main reason for Orca's limitation is due to its

unscrutinized interleaving of the DRL control and the classic Cubic control. Specifically, the RL part may undermine the theoretical guarantees of fairness provided by Cubic, while the Cubic part can affect the performance of the RL model by changing the sending rate according to its own internal state.

5.1.2 RTT-Fairness. A notable aspect of Jury is its RTT-fairness, characterized by the independence of its convergence property from the actual RTTs of competing flows. To empirically investigate this, we establish a 100Mbps link and sequentially introduce five Jury flows with progressively increasing base RTTs (70ms, 110ms, 150ms, 190ms, and 210ms). Each flow is launched at 60-second intervals and ran for 300 seconds. The throughput and RTT dynamics of these flows are depicted in Fig. 8. Our observations reveal that Jury flows, despite having varied RTTs, almost equally share the bandwidth, demonstrating negligible latency inflation. The reason is that Jury's bandwidth occupancy ratio estimation based on action-feedback signals is independent of network RTT. Consequently, Jury exhibits a convergence behavior that is adaptable to heterogeneous RTT scenarios. This adaptability ensures equal bandwidth sharing among flows originating from different network paths, further underscoring the robustness of Jury in diverse network conditions.

5.1.3 Large number of flows. In this section, we explore the scalability of Jury's fairness across large numbers of flows with varying running times and RTTs. We consider two common scenarios in the Internet: (i) flows with diverse running times, and (ii) flows with heterogeneous RTTs. For (i), we initialize a fixed number (4) of long-running flows that persist throughout the trial, along with a large number of short flows that arrive and depart frequently. The short flows follow an exponential distribution with $\lambda = 4$, and their running times are drawn from a Gaussian distribution $\mathcal{N}(4, 1^2)$. For (ii), we set up 20 competing flows, where half have a base RTT of 30ms and the other half have a base RTT of 90ms. We run each trace for 100 seconds and repeat the experiment 20 times. The average throughput and latency for both types of flows, as well as the overall throughput, are reported in Table 3. We find that Jury achieves similar throughput for heterogeneous flows in terms of RTT and running time, demonstrating its scalability and robust fairness across flows with diverse characteristics.

5.1.4 Friendliness. In this section, we further investigate the TCP friendliness of Jury, showing how Jury competes with flows with different CC policy (Cubic) and flow sizes. We set up a 100Mbps link with 30ms RTT and a buffer size of one BDP. In this experiment, a Jury flow and a Cubic flow are run concurrently for 120 seconds, and their throughput ratio is recorded. This test is repeated under various RTTs, with the results illustrated in Fig. 9. A ratio of 1 indicates ideal friendliness. Our observations show that while Jury attains higher throughput ratios compared to other learning-based schemes such as Aurora, Astraea, Orca, and Vivace, it remains more conservative than Cubic. Despite the good results, we note that Jury can only guarantee the convergence property across homogeneous flows with same CC algorithm, and there is no guarantee that the learned friendliness can be generalized to other network conditions. In fact, whether we need the generalizability of friendliness is an open question, as Cubic's link utilization explicitly degrades in lossy link. In this case, Jury will also deliver poor performance if we strictly follow the friendliness principle.

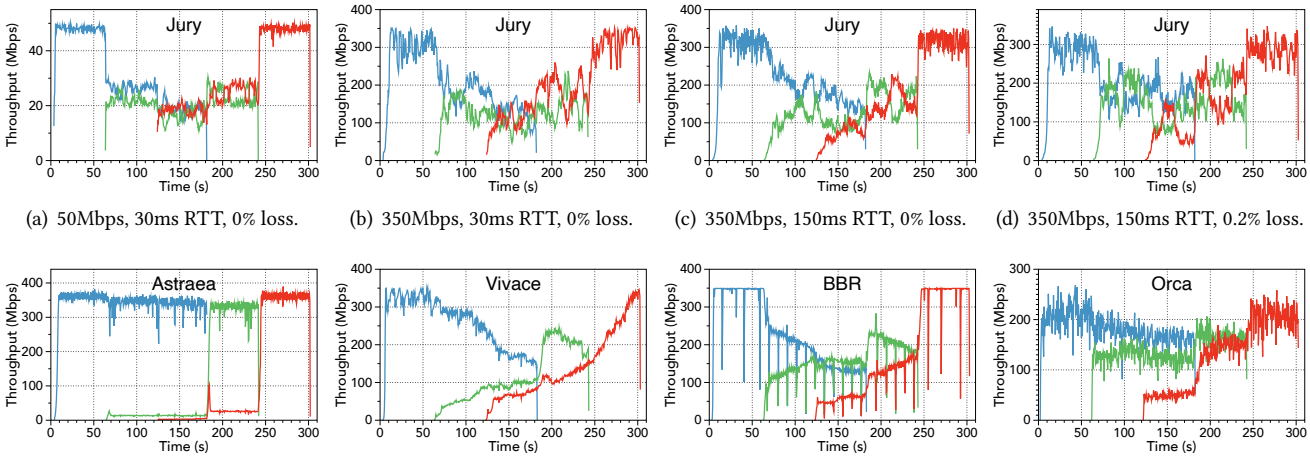
5.2 Consistent Performance

We further show Jury's performance generalizability by demonstrating its consistent high performance across a wide range of network environments through emulated experiments and real-world evaluations. We focus on improving the performance metrics (throughput, delay, and loss) of individual flows. While fairness is embedded in Jury through its pre- and post-processing pipelines, this design choice may limit the performance in certain scenarios where fairness considerations could affect the performance. For instance, larger flows may benefit from higher sending rates to achieve good performance, as seen in co-flow scheduling [5].

5.2.1 Extensive Emulations. To evaluate Jury's performance across various network environments, we set up a dumbbell topology with a single flow and varied link characteristics including link capacity, base delay, random loss rate, and queue buffer size. Subsequently, we individually alter each link characteristic, keeping others constant, to evaluate Jury and baseline systems. The parameters range from 10 to 600Mbps for bandwidth, 15 to 120ms for base delay, 0% to 1.5% for loss rate, and 0.2× to 16× BDP for buffer size.

The average results from 10 tests are displayed in Fig. 10 and the variances are within ±5%. For clarity, we exclude i) the latency results of Cubic in Fig. 10(f) and Fig. 10(h) as they are too large (>200ms) compared to other schemes, and ii) the link utilization results of Vegas in Fig. 10(a) as it is much lower when the link capacity is large (<0.3). We observe that, Jury, trained in a small region, consistently exhibits high link utilization and low latency inflation across all scenarios. We attribute Jury's consistent high performance to the invariance of input signals ensured by the signal transformation block, enabling performance transfer to unseen network conditions. In links with higher delays in Fig. 10(f), Jury exhibits small latency inflation (from 3.5ms to 7.2ms when increasing the one-way base delay from 15ms to 120ms), due to delayed network feedback.

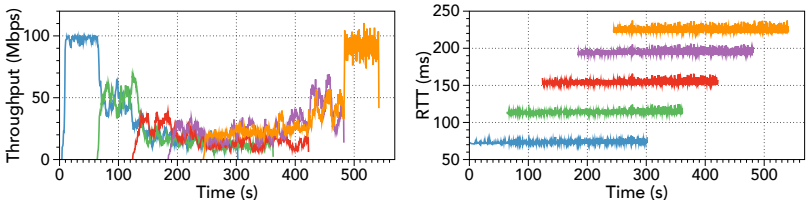
Contrastingly, DRL-based schemes like Aurora, Astraea and Orca, trained on a small training region demonstrate performance degradation when tested network environment configurations deviate from their training regions. For example, Aurora and Orca suffer from link under-utilization



(a) 50Mbps, 30ms RTT, 0% loss. (b) 350Mbps, 30ms RTT, 0% loss. (c) 350Mbps, 150ms RTT, 0% loss. (d) 350Mbps, 150ms RTT, 0.2% loss.

(e) 350Mbps, 30ms RTT, 0% loss. (f) 350Mbps, 150ms RTT, 0% loss. (g) 350Mbps, 150ms RTT, 0.2% loss. (h) 350Mbps, 150ms RTT, 0.2% loss.

Figure 7. Jury generalizes its fairness to network environments with different bandwidths, RTTs, and loss rates (Fig. 7(a),7(b),7(c),7(d)). On the contrary, previous learning-based or heuristic-based schemes fail to converge or converge slowly in some specific network conditions (Fig. 7(e),7(f),7(g),7(h)).



(a) Throughput (b) RTT

Figure 8. The throughput and RTT dynamics of multiple Jury flows with different RTTs.

Flows	Jury	
	Thr. (Mbps)	Delay ratio
Overall	192.3	1.64
Per Long flow	11.4	1.72
Per Short flow	10.9	1.88
Overall	191.7	/
Per small-RTT flow	10.3	1.72
Per large-RTT flow	11.1	1.27

Table 3. Long-short and heterogeneous RTT experiments.

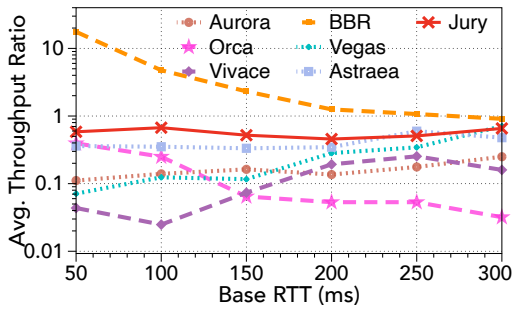
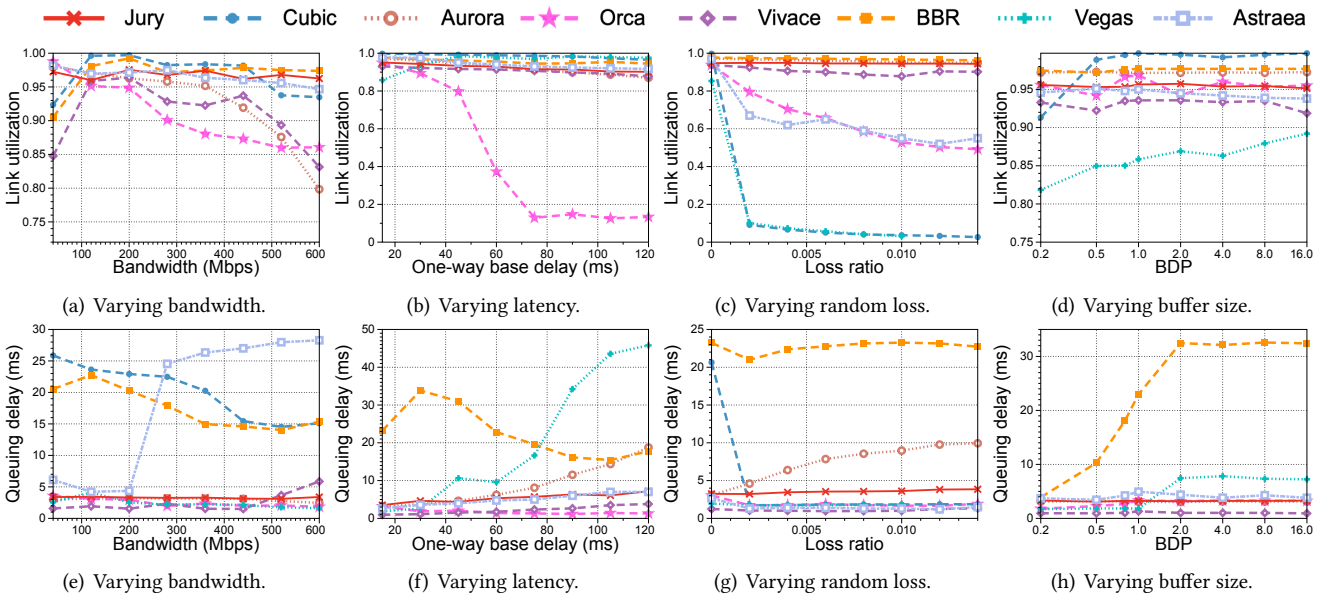


Figure 9. The throughput ratios of CC schemes to Cubic under varying RTTs.

when the link capacity is larger than 300Mbps (Fig. 10(a). Astraea shows severe latency inflation when the link bandwidth is large (Fig. 10(e)) and under-utilization when the link is lossy (Fig. 10(c)). Besides, Aurora shows proportional latency inflation with the increase of delay and loss rate, and Orca's link utilization dramatically drops (<20%) when the base delay of the evaluated scenario exceeds its training range (Fig. 10(f) and Fig. 10(g)). Their poor generalizability stems from input state and transition divergences under different network conditions. Additionally, Orca's reliance on Cubic leads to under-utilization when the random loss rate is large (Fig. 10(c)), as Cubic cannot differentiate between congestion and non-congestion loss.

5.2.2 Jury on Challenging Conditions. We extend our evaluation of Jury to more challenging network conditions: i) satellite links with large RTT; ii) high-speed links; and iii) LTE networks with rapidly fluctuating bandwidth.

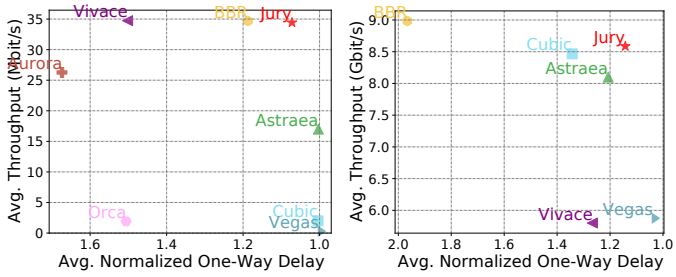
In satellite communication, long RTT and random loss significantly impact classical CC schemes. We test Jury on a simulated satellite link, following the setup from [8] with 42 Mbps bandwidth, 800ms RTT, and a 0.74% random loss rate. The results, averaged from 10 trials and shown in Fig. 11(a), reveal that Jury achieves over 75% link capacity and less than 5% latency inflation of the base one-way delay (18.2ms/400ms). This performance is attributed to its resilience to both high base delay and random loss. In contrast, Aurora, Astraea and Orca experience significant latency inflation or link under-utilization, aligning with the results in §5.2.1. Vivace incurs high latency inflation in satellite links. The reason is that



(a) Varying bandwidth. (b) Varying latency. (c) Varying random loss. (d) Varying buffer size.

(e) Varying bandwidth. (f) Varying latency. (g) Varying random loss. (h) Varying buffer size.

Figure 10. The performance of Jury and baselines in terms of link utilization and queuing delay under various bandwidths, base delays, loss rates, and bottleneck buffer sizes.



(a) Satellite network. (b) High-speed network.

Figure 11. Jury in various network conditions.

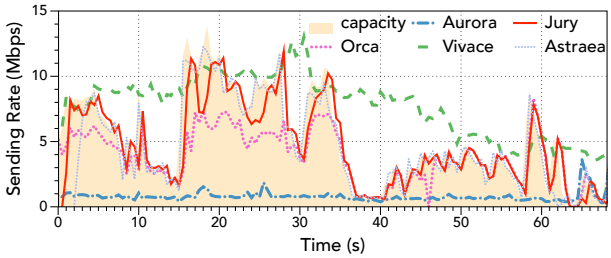


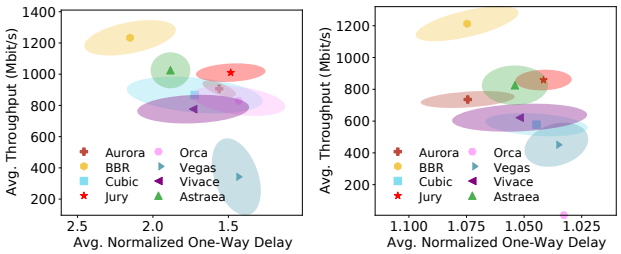
Figure 12. The responsiveness in LTE networks.

before achieving the stable equilibrium operating point, Vivace's RTT-based control frequency incurs large latency and packet loss during its slow convergence progress when the RTT is much higher. For high-speed networks, we set up a 10 Gbps connection with 15ms latency in our testbed to mimic real-world WAN conditions. Fig. 11(b) displays the throughput and latency for Jury and benchmarks. We observe that Jury demonstrates link utilization comparable to BBR, but with lower latency, showcasing its strong adaptability to high-speed networks which are far beyond its training bandwidths.

Similar with [23], We also assess Jury's responsiveness in an LTE network environment [44] by emulating realistic cellular network traces with fluctuating bandwidth. Fig. 12 shows that Jury, with a 15ms base one-way delay and adequate buffering, responds excellently to the dynamic bandwidth changes, outperforming previous learning-based models. On the other hand, Aurora, while managing acceptable link utilization under lower bandwidth scenarios (Fig. 10(a)), struggled in the LTE link where bandwidth fell outside its training scope (approximately 5 Mbps). Additionally, Vivace could hardly react to the dynamic changing bandwidth due to its slow convergence speed. Jury's good responsiveness is due to the effective balance between exploration and exploitation in its interval-based rate adjustment, learned adaptively through DRL.

5.2.3 Real-world Experiments.

We conduct real-world experiments to assess Jury's performance on the wild Internet, using the AWS platform with c5a.4xlarge instances in Seoul, Tokyo, and London. The sender is located in Seoul, while the receivers are alternated between Tokyo and London, allowing us to evaluate Jury in both intra- and inter-continental environments. We run each CC algorithm for 60 seconds in each test, and repeat the test 10 times. The average results are displayed in Fig. 13.



(a) Intra-Continental Experiments (b) Inter-Continental Experiments

Figure 13. Throughput and one-way delay in real-world testbed.

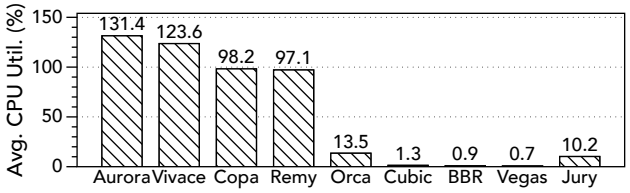


Figure 14. Average CPU utilization of Jury and CC baselines.

Our observations indicate that Jury achieves high link utilization and maintains low latency inflation across both intra-continental and inter-continental links, forming a new Pareto frontier among the baselines. Notably, Jury delivers higher throughput and lower latency than Cubic in both scenarios. The fact that Jury excels in real-world Internet conditions, despite being trained on a small region of network environments, underscores its theoretical generalizability. In contrast, as previously discussed in §2, Aurora exhibits higher latency inflation, and Orca struggles with large BDP links in the inter-continental experiments, highlighting their generalizability issues. BBR achieves higher throughput at the cost of latency inflation due to i) its aggressiveness policy and ii) explicit modeling policers on the Internet, which is out of the scope of this paper.

5.3 Overhead

CPU utilization. We evaluate the computational overhead of Jury and other CC baselines by measuring their CPU utilization during a flow's transmission. Specifically, we emulate a link with 100 Mbps bandwidth, 30 ms RTT, and a buffer size of 1 bandwidth-delay product (BDP). We configure Jury to infer the new sending rate every 20 ms, aligning with the update frequency of the previous DRL-based algorithm Orca [1]. For each CC algorithm, we run the flow transmission for 120 seconds and record the CPU utilization of the transmission process. During the transmission, Jury takes an average of 4.5 ms for each model invocation. This low latency for the inference process is crucial for the real-time performance of the congestion control algorithm. Choosing a time interval smaller than it may lead to system failure.

The CPU utilization results are then averaged and shown in Figure 14. The results indicate that Jury achieves lower computation overhead than Orca due to its efficient C++ implementation. We further observe there is almost no difference between the overhead results for Jury with or without the post-processing function, as the computation cost of the post-processing is much lower compared to the DRL component. To further reduce the computation overhead of DRL-based CC algorithms, we can either reduce the inference frequency [41] or implement the neural network model in the kernel mode [47], which is out of the scope of this paper.

6 Related Work

Congestion Control The congestion control task has been a persistent hotspot in networking research for more than three decades. The traditional schemes [2, 3, 13, 15, 17] are frequently referred to as heuristic-based schemes since they are typically handcrafted based on specific assumptions of the network condition. For instance, loss-based protocols [15, 17] adopt packet loss as the congestion signal and respond to it by reducing the congestion window.

Recently, a plethora of learning-based schemes has been proposed to learn control policy based on data instead of using predetermined rules [1, 7, 8, 18, 24, 36, 41, 43, 45]. For example, PCC Allegro [7], Vivace [8], and Libra [9] utilize a stateless online learning paradigm to explore the best action through trial and error. Aurora [18] proposes to adopt vanilla deep reinforcement learning on congestion control to adjust the *cwnd*, which raises several challenges such as fairness and overhead issues. Orca [1] proposes to resolve these challenges by combining classic CC schemes (*e.g.*, Cubic) with DRL-based model. Astraea [23] introduces a global fairness metric into its multi-agent reinforcement learning framework to improve convergence properties. We have thoroughly compared Jury with these prior schemes in terms of fairness generalizability in both the motivation and evaluation sections. We refer readers to [19] for a comprehensive survey of learning-based CC schemes.

RL Generalization Having achieved superior performance on benchmarks such as Atari [31] and MuJoCo [42] that consist of a single environment for both training and evaluation, the RL community has started to focus on understanding, measuring, and improving generalizability in DRL [6, 20, 25, 33, 35, 39]. Among them are works proposing new architectures or training paradigms to improve DRL generalizability. For example, [35] adopts an automatic data augmentation technique to improve the sample complexity and RL generalizability. [39] introduces training environment diversity by

inviting multiple agents into the environment so that the RL agent can learn a policy with better generality. [21] gives a detailed survey of RL generalization problems and solutions.

However, there is no general generalizability guarantee of an RL model [20], as the difference between the training and test environments is task-related. Hence, to improve the generalizability of a DRL-based CC scheme, a domain-specific analysis is demanded, which is the focus of our work.

7 Conclusion

In this paper, we presented Jury, a DRL-based CC algorithm that achieves fairness generalizability across various network environments while maintaining high performance. Jury establishes robust fairness generalizability theoretically and practically. Extensive evaluations demonstrate consistent fairness and performance in both emulated and real-world Internet settings. The convergence analysis and innovative decoupling technique presented in this paper, we believe, will shed light on the design of future learning-based algorithms in various networking and system applications that demand specific properties.

Acknowledgments

We thank the anonymous reviewers and our shepherd Zsolt István for their constructive comments. This work is supported in part by the Hong Kong RGC TRS T41-603/20R, GRF 16213621, NSFC 62062005, 62402407. Kai Chen is the corresponding author.

References

[1] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: a pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 632–647.

[2] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 329–342.

[3] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. 1994. *TCP Vegas: New techniques for congestion detection and avoidance*. Number 4. ACM.

[4] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *Queue* 14, 5 (2016), 20–53.

[5] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 443–454. https://doi.org/10.1145/2619239.2626315

[6] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 1282–1289.

[7] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 395–408.

[8] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 343–356.

[9] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. 2021. A unified congestion control framework for diverse application preferences and network conditions. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 282–296.

[10] Salma Emara, Fei Wang, Baochun Li, and Timothy Zeyl. 2022. Pareto: Fair congestion control with online reinforcement learning. *IEEE Transactions on Network Science and Engineering* 9, 5 (2022), 3731–3748.

[11] DI engine Contributors. 2021. DI-engine: OpenDILab Decision Intelligence Engine. https://github.com/opendilab/DI-engine.

[12] Eyal Even-Dar, Yishay Mansour, and Uri Nadav. 2009. On the convergence of regret minimization dynamics in concave games. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 523–532.

[13] Sally Floyd, Tom Henderson, Andrei Gurtov, et al. 1999. The NewReno modification to TCP's fast recovery algorithm. (1999).

[14] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 1587–1596.

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

[16] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. 2020. {LinnOS}: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 173–190.

[17] Van Jacobson. 1988. Congestion avoidance and control. *ACM SIGCOMM computer communication review* 18, 4 (1988), 314–329.

[18] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *International Conference on Machine Learning ICML*. 3050–3059.

[19] Huiling Jiang, Qing Li, Yong Jiang, Gengbiao Shen, Richard Sinnott, Chen Tian, and Mingwei Xu. 2021. When machine learning meets congestion control: A survey and comparison. *Computer Networks* 192 (2021), 108033.

[20] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. 2021. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794* (2021).

[21] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. 2021. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794* (2021).

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[23] Xudong Liao, Han Tian, Chaoliang Zeng, Xinchen Wan, and Kai Chen. 2024. Astraea: Towards Fair and Efficient Learning-based Congestion Control. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 99–114.

[24] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. 2022. Multi-objective congestion control. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 218–235.

[25] Dhruv Malik, Yuanzhi Li, and Pradeep Ravikumar. 2021. When Is Generalizable Reinforcement Learning Tractable? *Advances in Neural Information Processing Systems* 34 (2021), 8032–8045.

[26] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 197–210.

[27] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 270–288. https://doi.org/10.1145/3341302.3342080

[28] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC proteus: Scavenger transport and beyond. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 615–631.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[32] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 417–429.

[33] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720* (2018).

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[35] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. 2020. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862* (2020).

[36] Alessio Sacco, Matteo Flocco, Flavio Esposito, and Guido Marchetto. 2021. Owl: congestion control with partially invisible networks via reinforcement learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.

[37] J Salim, H Khosravi, Andi Kleen, and Alexey Kuznetsov. 2003. *Linux netlink as an ip services protocol*. Technical Report.

[38] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[39] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. 2021. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808* (2021).

[40] The Google BBR Team. [n. d.]. BBR bandwidth based convergence. https://github.com/google/bbr/blob/master/Documentation/bbr_bandwidth_based_convergence.pdf.

[41] Han Tian, Xudong Liao, Chaoliang Zeng, Junxue Zhang, and Kai Chen. 2022. Spine: an efficient DRL-based congestion control with ultra-low overhead. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 261–275.

[42] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 5026–5033.

[43] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-Generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 123–134. https://doi.org/10.1145/2486001.2486020

[44] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 459–471.

[45] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*.

[46] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. 2021. ACC: Automatic ECN Tuning for High-Speed Datacenter Networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 384–397. https://doi.org/10.1145/3452296.3472927

[47] Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen. 2022. Liteflow: towards high-performance adaptive neural networks for kernel datapath. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 414–427.