Astraea: Towards Fair and Efficient Learning-based Congestion Control

Xudong Liao*, Han Tian*, Chaoliang Zeng, Xinchen Wan, Kai Chen
iSING Lab, Hong Kong University of Science and Technology

Abstract

Recent years have witnessed a plethora of learning-based solutions for congestion control (CC) that demonstrate better performance over traditional TCP schemes. However, they fail to provide consistently good convergence properties, including *fairness*, *fast convergence* and *stability*, due to the mismatch between their objective functions and these properties. Despite being intuitive, integrating these properties into existing learning-based CC is challenging, because: 1) their training environments are designed for the performance optimization of single flow but incapable of cooperative multi-flow optimization, and 2) there is no directly measurable metric to represent these properties into the training objective function.

We present Astraea, a new learning-based congestion control that ensures fast convergence to fairness with stability. At the heart of Astraea is a multi-agent deep reinforcement learning framework that explicitly optimizes these convergence properties during the training process by enabling the learning of interactive policy between multiple competing flows, while maintaining high performance. We further build a faithful multi-flow environment that emulates the competing behaviors of concurrent flows, explicitly expressing convergence properties to enable their optimization during training. We have fully implemented Astraea and our comprehensive experiments show that Astraea can quickly converge to fairness point and exhibit better stability than its counterparts. For example, Astraea achieves near-optimal bandwidth sharing (i.e., fairness) when multiple flows compete for the same bottleneck, delivers up to 8.4× faster convergence speed and 2.8× smaller throughput deviation, while achieving comparable or even better performance over prior solutions.

*Equal contribution.

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

Keywords: Congestion Control, Reinforcement Learning, Transport Protocol

ACM Reference Format:
Xudong Liao, Han Tian, Chaoliang Zeng, Xinchen Wan, Kai Chen. 2024. Astraea: Towards Fair and Efficient Learning-based Congestion Control. In *Nineteenth European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3627703.3650069

1 Introduction

Internet congestion control (CC) remains an active field of research in both academia and industry. An expected end-to-end Internet CC algorithm should preserve two abilities. First, the algorithm should achieve high performance, i.e., high throughput, low latency, and few congestion loss in various network conditions. Second, it needs to provide good convergence properties, i.e., rate allocation in a fair and efficient manner[1], and fast convergence with stability. Despite three decades of efforts, it is still hard to achieve the above requirements simultaneously.

Classical TCP CC algorithms [6, 7, 14, 16, 19, 20, 36] have been notorious for performance degradation when their assumptions about congestion and packet-level events are violated [10, 34]. To address this problem, a recent evolved thread of research has provided us with a plethora of clean-slate learning-based CC approaches [3, 10, 11, 18, 26, 33, 37, 40]. These learning-based schemes achieve performance goals by defining an objective function consisting of throughput, latency, and loss rate. Offline training [3, 18] or online optimization [11] are then applied to guide the rate control. Learning-based approaches enable the potential to adapt CC to various network conditions and hence achieve consistently high performance.

While substantially improving over traditional TCPs in terms of performance, these learning-based CC schemes have so far shown little improvement on convergence properties. For example, as shown in §2, Vivace [11], an online learning-based scheme, shows a hard tradeoff between convergence speed and stability, although it is designed with these metrics in mind. Moreover, Aurora [18], a reinforcement learning

[1]The efficient manner means all participant flows should fully utilize the bottleneck capacity.

(RL)-based scheme, is fairness-agnostic and shares no bandwidth with competing flows. Orca [3] identifies the similar convergence issue, and proposes a coupling idea by incorporating classical TCP into RL-based control. Therefore, it can preserve certain fairness provided by TCP. However, based on our evaluations (§5.2), we find that Orca achieves unstable convergence. The reason behind is that Orca's RL part may hurt the theoretical guarantee of fairness from AIMD by suppressing the loss event.

The crux is that existing learning-based CC schemes *do not directly* optimize for the convergence properties, because their decentralized learning paradigms only optimize local performance objectives based on local observations. They try to reach the convergence indirectly by optimizing for the performance goals, which often turns out to be sub-optimal. For instance, Vivace empirically maximizes a latency-sensitive objective and guarantees a fair equilibrium when competing flows follow the same optimization method. It, however, may converge slowly and lead to an inefficiently fair allocation in the wild Internet, which was uncovered by [3]. Moreover, it is difficult to tune the control knobs to achieve a good tradeoff between performance goals and various convergence characteristics, as shown in §2, because these knobs are generally not related to the convergence properties semantically.

Therefore, we ask: *is it possible to automatically develop a CC algorithm that can quickly converge to fair rate allocation with stability, while maintaining high throughput, low latency, and low packet loss rate?*

In this paper, we answer this question affirmatively with Astraea. Astraea is an end-to-end RL-based CC scheme for Internet that *explicitly* introduces convergence metrics including fairness, convergence speed, and stability, in addition to throughput, latency, and loss, into the optimization objective. Despite being intuitive, Astraea calls for a fundamental change in the training framework design compared with prior learning-based CC systems: Astraea adopts a centralized and cooperative multi-agent training paradigm considering the dynamics between multiple flows to optimize global performance.

In Astraea, each flow is guided by an RL *agent*, which perceives packet statistics as the input of its control policy, and enforces back the new *cwnd*. During the training, Astraea collects observations from all concurrent participants in the network, and issues a reward signal encoding the performance and convergence metrics to the RL policy for reinforcement. Eventually, through joint optimization on these goals, the RL agent learns a control policy that works collaboratively with competing flows to achieve a fair, efficient and stable equilibrium without using any pre-defined control rules or hardwired modeling about the network.

Realizing such multi-agent DRL in CC, however, is challenging. First, before Astraea, there is no faithful training playground for CC that supports directly studying multiple competing flows. In order to enable explicit optimizations on the aforementioned metrics, we need to feed the training algorithm with the global information of all participant flows. We address this challenge by designing a novel multi-flow environment (§3.2). It can establish multiple concurrent flows in the network and provide measurements on convergence properties. Our environment supports flexible flow configurations to enable complex network conditions, simulating the dynamics of the wild Internet and performing as an effective training suite for Astraea algorithm.

Second, it is not easy to encode convergence properties into RL reward objective, as it lacks directly measurable metrics (like throughput and latency) to evaluate these properties. To solve this problem, we design novel representations of convergence properties, i.e., fairness and stability, with readily achievable average throughput of each flow (§3.3). As a result, we can simply represent the global reward function in a linear combination of metrics that we focus on, i.e., performance requirements and convergence properties.

Third, we find that training Astraea with classical DRL algorithms suffers from large variance, as the complexity of the environment increases with the number of concurrent flows in the bottleneck. To tackle this challenge, we leverage a customized multi-agent DRL training algorithm (§3.4), which incorporates observed empirical trajectories of all active flows and calculates the global reward as a signal. Compared with the standard RL training procedure, it introduces global information involving all active flows to reduce the estimation variance of the value-action function, which will stabilize and accelerate the multi-agent RL training.

We have implemented Astraea and corresponding environment, and performed efficient distributed training for speedup. We integrate the trained Astraea algorithm with Linux kernel TCP (§4). Extensive experiments (§5) over emulation and real-world Internet demonstrate that Astraea significantly improves the convergence properties while preserving high performance in a diverse range of network conditions and multiple bottleneck scenarios. For example, in the experiments of multiple homogeneous flows, Astraea shows near-optimal fairness, achieves the average Jain index of 0.991 and improves the convergence speed and stability by up to 8.4 × and 3.3× compared with existing TCP variants and learning-based schemes. In addition, Astraea delivers comparable RTT fairness and TCP friendliness with existing learning-based schemes. In real-world experiments, Astraea always defines the frontiers of high throughput and low latency: it delivers 3.1× higher throughput than Orca and 1.4 × lower latency than BBR.

In summary, we make the following contributions.

- We present Astraea, a multi-agent DRL solution for CC that aims to directly optimize for fairness, convergence speed and stability as well as the performance goals. Astraea is a showcase of solving distributed problems with centralized learning methods.

Algorithm	Fairness	Fast Convergence	Stability
Aurora [18]	✗	✓	✗
Vivace [11]	✓	✗	✗
Orca [3]	✓	✓	✗
Astraea	✓	✓	✓

Table 1. Comparison of learning-based algorithms. None of the existing learning-based algorithms can satisfy all three requirements. Note that Orca is coupled with CUBIC by default, which results in the issue of stability. Besides, as Orca's RL module does not consider optimizing for fairness, there is no effort on this metric beyond TCP.

- We design and implement a multi-flow environment for training Astraea. To the best of our knowledge, this is the first CC playground that implements state and action passing mechanism, and global information gathering for the multi-agent training algorithm. The environment will also facilitate the further study on convergence behaviors of heterogeneous CC schemes.
- We evaluate Astraea over real-world and emulated environments and show that Astraea significantly improves convergence properties while preserving high performance.

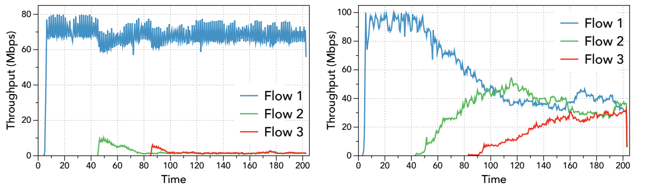
The code of Astraea is available at https://github.com/HKUST-SING/astraea.

2 Motivation

Driven by the tremendous successes achieved by machine learning in computer systems and networks [8, 27, 28], recent proposals seek to leverage learning-based methods to design efficient CC algorithms [3, 11, 18]. While these schemes have demonstrated superior performance over traditional CC algorithms by providing high throughput, low latency, and low loss rate, they fall short of achieving well-behaved convergence properties, i.e., *fast convergence to fairness with stability and efficiency*, as summarized in Table 1. All of them cannot fulfill fairness, stability and responsiveness simultaneously.

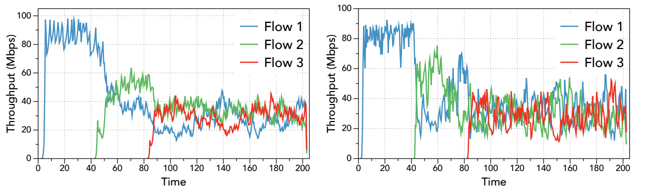
The reason for the failure, we argue, is that their predefined local utility functions are often hard to align with these desiderata, especially *global fairness*. In the following, we first give a brief introduction of two representative learning-based congestion control: Aurora [18] and Vivace [11], and exemplify the objective discrepancy by experimental results.

Aurora is among the first to introduce deep reinforcement learning (DRL) to solve congestion control problem, whose core is a carefully designed reward function (Equation 1) that encodes the network utility. It performs offline training to learn a control policy that maximizes the accumulated reward and uses this policy to direct the sending process. On the other side, Vivace defines a latency-aware utility function (Equation 2, x_i and L_i the sending rate and loss rate in monitor interval i, respectively.) and controls sending rate by performing gradient-ascent online learning to optimize this utility function. While achieving high performance, they both have different issues related to convergence properties.



(a) Aurora is very unfair. **(b)** Vivace converges slowly.

Figure 1. Existing learning-based algorithms fail to fast converge to fairness with stability.



(a) Vivace *could* converge quickly. **(b)** Vivace introduces instability.

Figure 2. Enhanced Vivace performs diversely.

$$r = 10 \times \text{throughput} - 1000 \times \text{latency} - 2000 \times \text{loss} \quad (1)$$

$$u = x_i^{0.9} - 900 \times x_i \frac{d(RTT_i)}{dT} - 11.25 \times x_i \times L_i \quad (2)$$

Aurora is unfair. We evaluate Aurora with multiple flows on a link with 80 Mbps bandwidth, 60 ms RTT, and enough buffer (4.8 MB) to absorb the traffic. As shown in Figure 1a, Aurora is so aggressive that it would not allow other flows to get any share of bandwidth. Aurora's reward function accounts for this behavior, as it emphasizes throughput, while being ignorant of achieving stability and equal-sharing of the network bandwidth.

Vivace shows slow responsiveness. We emulate a network with a bandwidth of 100 Mbps, 1 BDP buffer size, and vary the base RTT to evaluate Vivace's convergence. We start three flows in sequence, with a launching interval of 40 seconds. In the experiment of Figure 1b, the base RTT is set to be 120 ms. We find that Vivace can hardly achieve the fairness point before all flows terminate. The reason is that while Vivace has proof of fairness, its online learning-based rate control is based on trial-and-error, which costs several probing steps to find a better response and therefore leads to explicit inefficiency in large RTT networks.

Both Aurora and Vivace fail to align with global properties by optimizing the local objective function, which is problematic in the real-world network environment where bandwidth and RTT may vary considerably [7, 10]. For example, Vivace may waste the link capacity when transplanted to high-speed, e.g., 10Gbps networks.

Furthermore, improving performance by adjusting hyperparameters on these learning-based CC schemes often leads to degradation when the network environment changes and the *adjustments* become invalid again, as the mapping between the local utility function and the global goal keeps changing according to the network dynamics. In what follows, we showcase the experiment of tuning parameters in Vivace.

We attempt to improve Vivace's responsiveness by tuning its *initial conversion factor*, letting Vivace put more rate increment on each probing step ($r_{new} = r + \theta_0 \gamma$) to converge more quickly. We properly enlarge θ_0 and the result is presented in Figure 2a. Now Vivace becomes more responsive to other flows and can quickly lead to a fair convergence. It *seems* that our adjustment aligns Vivace's local objective with global properties closer. However, as shown in Figure 2b, this will cause much instability when the *enhanced* Vivace is evaluated in a network environment of 12ms RTT, where it can even hardly achieve a convergence!

To this end, we argue that existing practice of achieving these goals by optimizing local objectives is intrinsically questionable. Therefore, instead of performing the daunting work for tuning local objectives, we advocate *optimizing global goals directly* for agile responsiveness to network dynamics and fast convergence to a stable and efficient state, motivating the design of ASTRAEA.

3 Design

3.1 Overview

To achieve *explicit* optimization on convergence properties, we design ASTRAEA to directly fulfill the insight derived in §2. The major advantages of ASTRAEA come from the multi-flow training environment and the multi-agent RL algorithm that directly rewards fast convergence to fairness with stability.

Architecture: Figure 3 overviews ASTRAEA, which consists of three main components: a multi-flow training environment (§3.2), RL agents which execute control policy (§3.3), and a Learner which updates the policy using multi-agent RL training algorithm (§3.4).

Basically, ASTRAEA's training environment establishes the network with multiple concurrent flows. It contains three modules: a Flow Generator, a Runtime and a Controller. With user-defined configurations, the Flow Generator starts flows runtime with pre-defined characteristics. The Controller exchanges necessary information between running flows and RL agents. Based on these modules, RL agents and the Learner in ASTRAEA perform the multi-agent RL training algorithm. RL agents conduct congestion control for each flow through mapping states to actions, and the Learner collects experiences from RL agents and global information from the controller as training data to refine the policy.

Workflow: Generally, ASTRAEA works as follows. It launches RL agents for each active flow to direct the sending behavior via exchanging information with the Controller. During

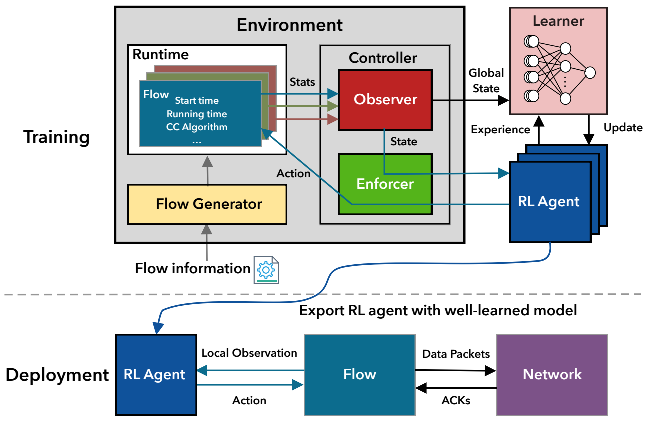


Figure 3. ASTRAEA Framework.

training, all RL agents are initialized with a shared control policy from the centralized Learner. Each flow in the Runtime collects packet-level statistics at the granularity of Monitoring Time Period (MTP) [3] and launches an action request to the Controller with these statistics information (local state in §3.3) as arguments to get the new *cwnd*. Upon receiving a request from one flow, the Observer relays the request to the Enforcer, which then forwards the request to the corresponding RL agent. At the same time, the Observer collects packet-level statistics (local state) from all other active flows to compile the global state (Table 2) for the Learner. After receiving the requests from the Enforcer, the RL agents respond to it with new *cwnd* based on the control policy, and generate interaction trajectories between flows and the Environment. The trajectories, also known as experiences in the RL area, fuel up the following RL training. Altogether, with the global state and the trajectories provided by RL agents, the Learner issues a global reward encoding performance and convergence metrics and trains its policy to maximize the reward objective. The updated policy is periodically pushed back to each RL agent.

Evaluation: In the deployment phase, as shown in Figure 3, each flow loads one RL agent with the well-learned policy from Learner to perform congestion control. Flow sender gathers local observations from each ACK and relays the information to the corresponding RL agent for further control adjustment. We note that global information is only utilized during the training process to guide the learning, and is not included as part of the model input. Therefore, the policy execution of ASTRAEA does not necessitate global information during the deployment phase.

3.2 Environment

In general, ASTRAEA environment enables the competing of multiple flows in the network and provides the interface to obtain the global states from all active participants, which is the key difference between our environment and prior solutions. Training environments in previous work only provide local information about one single flow. Therefore, they can only support direct optimization on performance metrics of throughput, latency and loss. In contrast, Astraea environment sheds the light on the explicit study of global objectives, e.g., convergence properties, by inferring from the global states.

As shown in Figure 3, within the Environment, the Runtime module consists of an emulated link with multiple concurrent flows. The link is set to simulate the network bottleneck, which is able to control the packet ingress rate and egress rate to simulate specific bandwidths and delay each packet for a certain time to simulate the base round-trip time. The emulated link can also configure user-defined queuing policies and perform randomized packet dropping to simulate non-congestion packet loss. In what follows, we demonstrate the functionalities of other modules in the multi-flow environment and illustrate the support for multi-agent training.

Flow generator. The Flow Generator in Astraea environment starts network flows according to user-defined configuration, such as flow starting time, running time, the congestion control algorithm, and extra delay for specific flows. We introduce randomization into flow starting points and running time to emulate the dynamics of the Internet and thus enhance our environment's fidelity. In particular, we recommend modeling flow arrivals at bottleneck as Poisson arrivals [13] rather than deterministic arrivals, which helps RL algorithms obviate from overfitting on specific deterministic traffic patterns of network flows. The flexible configuration in our environment allows for the training and evaluation of different metrics. For instance, we can study whether flows share the bandwidth equally, by using the same CC scheme and the same base RTT for all flows (which we call homogeneous flows); also, by using various CC schemes or heterogeneous RTTs, we can study the friendliness or RTT fairness [29] respectively.

Flow-driven control paradigm. In a typical RL problem, the interaction between the environment and the agent is driven by the latter, which means that the RL agent proactively observes the environment, derives action from observation, and enforces the action back. However, in the network field, it's more natural and flexible to trigger the state-action control logic from the network flow side. The reason is that the congestion control logic of each flow is asynchronously triggered by packet-level events or MTP expiration, which will introduce high control complexity in the agent-driven setting. Thus we design flow-side triggering where network flows proactively request control action after collecting adequate packet statistics for each MTP, making our workflow easily scalable with multiple RL agents.

Observer and Enforcer. Controller in the environment consists of an Observer and an Enforcer. In the multi-agent RL training scenario, we need to feed the RL algorithm with observations of all active flows (we call them world observations) in the Runtime. To this end, we dedicate a centralized Observer for this purpose. When each flow sends its local packet-level statistics to the Observer to get the control action, world observation signals will be sent from the Observer to other active flows. Upon receiving signals, active flows will issue a response containing their own observations. By stacking these observations in responses, Observer then forms a global state describing the current status of the Runtime, which will be sent to the Learner for training. Enforcer performs as a proxy between the environment and RL agents: it passes the state from each flow and applies action back.

The following subsections present the design details of how Astraea leverages information from the multi-flow environment to design the RL agent and the training algorithm.

3.3 RL Agent

The RL agent decides the congestion control policy of each flow. For each monitoring time period (MTP), the agent perceives the packet statistics from the network environment and generates a new *cwnd* for the next MTP to maximize the target performance goal. It has three key building blocks: the state block generating flow state from the world observation, the action block updating *cwnd*, and the reward block defining the performance goal. Among them, the reward block is the most challenging one, since it is difficult to measure convergence properties directly. We address this issue by carefully designing a suitable representation of convergence properties from the average throughput of each participant flow, which is readily computable and achievable. Besides, Astraea's reward design differs from previous works [3, 18] by considering the packet statistics of all active flows in a centralized style, thus enabling expressing global subgoals in the network. During the training, the agent is informed with global information from these blocks to learn the optimal interaction strategy between multiple flows. We describe these blocks in detail.

State block. Upon receiving the packet statistics of incoming ACKs in the last MTP from the world observer, the state block assembles the local and global states for the training and inference of RL agents. The local state contains collectible statistics on end-hosts, and the global state contains global statistics across all active flows in the network. The state block works in two modes. In the training mode, it receives the packet statistics from all active flows in the environment, assembles both local and global states of the flow agent, and performs the multi-agent RL training algorithm. In the inference mode, only the local state is generated based on the flow information of its own and used for inference. We run the training mode for model learning and the inference mode for evaluation on the fly.

For packet statistics received by a flow agent, we denote the throughput as thr, latency as lat, lost bytes as $loss$, the number of packets in flight as pkt_{flight} and the pacing rate as p_{rate}. Our RL agent utilizes the following statistics in a MTP for the local state.

- The throughput ratio, the ratio of the average throughput to the maximum throughput in the flow's history $\frac{thr}{thr_{max}}$.
- The maximum throughput thr_{max} in the flow's history.
- The latency ratio, the ratio of the average latency to the minimum latency in the flow's history $\frac{lat}{lat_{min}}$.
- The minimum latency lat_{min} in the flow's history.
- The relative congestion control window, the current congestion control window $cwnd$ divided by the maximum throughput and the minimum latency: $\frac{cwnd}{thr_{max} * lat_{min}}$.
- The ratio of average loss rate to the maximum throughput $\frac{loss}{thr_{max}}$.
- The ratio of the packets in flight to the current $\frac{pkt_{flight}}{cwnd}$.
- The ratio of the pacing rate to the maximum throughput in the flow's history $\frac{p_{rate}}{thr_{max}}$.

In order to generalize to different network conditions and accelerate training, we normalize all the features in the local state, except for the maximum throughput and minimum latency, so that the agent will have similar states that are independent of network conditions. Besides, the maximum throughput and minimum latency features will help the agent make discriminative decisions according to the network characteristics. For example, the network with high RTT links tends to respond to the agent's action more slowly, thus calling for a more conservative sending rate policy to avoid bufferbloat.

For the global state, instead of directly concatenating the local states of all active flows together, the state block performs aggregation on the local statistics vectors to reduce the input feature dimension, leading to more effective training. We calculate the minimum, maximum, and average metrics to help the agent estimate the fairness across flows. We also collect the link information, including the delay, buffer size, and bandwidth, to enable a better value estimation of the current state for the training algorithm in Section 3.4. The full definition of global state information is shown in Table 2. These values will be used as the part of input for the critic in the training algorithm for better value estimation.

To provide sufficient information for the agent to make the proper decision, the state block stacks a fixed-length (denoted by w) history of the per-MTP state as the final input state for the RL model, which consists of the network statistics in the last w MTPs.

Action block. The action block returns the $cwnd$ for one flow to conduct congestion control. The action block feeds the RL model with the input state from the state block and gets the output action a, which is in the range $(-1, 1)$. The output action a is then transformed to obtain the $cwnd$ for the next MTP. We use the same mapping function between the $cwnd$ and a as that in Aurora [18], as it controls $cwnd$ robustly and stably. With the present $cwnd_t$ and the output action a_t in the t-th MTP, the action block calculates the $cwnd$ for the next MTP as follows:

ovr_thr	The overall throughput of all active flows.
min_thr	The minimum current throughput of all active flows.
max_thr	The maximum current throughput of all active flows.
avg_lat	The average latency of all active flows.
min_cwnd	The minimum current $cwnd$ of all active flows.
max_cwnd	The maximum current $cwnd$ of all active flows.
avg_cwnd	The average current $cwnd$ of all active flows.
$loss\_ratio$	The average loss ratio of all active flows.
num_flow	The number of active flows.
d_0	The base one-way-delay of the link.
buf	The buffer size.
c	The bandwidth of the link.

Table 2. Global state information generating by the state block.

$$cwnd_{t+1} = \begin{cases} cwnd_t * (1 + \alpha a_t) & a_t \geq 0 \\ cwnd_t / (1 - \alpha a_t) & o.w. \end{cases} \tag{3}$$

Based on the updated $cwnd$, the pacing rate is obtained by dividing the present $cwnd$ by the smoothed RTT of packets $\frac{cwnd}{sRTT}$. In the equation 3, the coefficient α controls the responsiveness of the agent. With a larger α, the agent is able to exploit a larger range near the present $cwnd$ in a single MTP, which, however, may also cause an unstable sending rate and network fluctuations.

Reward block. The reward block is the core building block of Astraea, which defines our global goal of congestion control and works during the learning process. Specifically, we design the reward function R as a linear combination of metrics reflecting our congestion control subgoals, including efficiency, stability, fairness, and responsiveness.

Assume that there are n active flows in the link. Fed with packet statistics, the reward block calculates each metric as follows. We employ the throughput metric as the ratio of the present overall throughput to the link bandwidth and the loss metric as the average ratio of the loss rate to the current throughput across flows:

$$R_{thr} = \frac{\sum_i thr_i}{c} \qquad R_{loss} = \frac{1}{n} \sum_i \frac{loss_i}{thr_i} \tag{4}$$

For the latency metric, [3] proposed to ignore small queuing delay to allow flows to reach the maximum throughput in dynamic links. We employ this idea and design our latency metric as the following:

$$R_{lat} = \begin{cases} (\frac{1}{n} \sum_i lat_i - (1 + \beta) d_0) p_{rate} & \frac{1}{n} \sum_i lat_i > (1 + \beta) d_0 \\ 0 & o.w. \end{cases} \tag{5}$$

where d_0 denotes the base delay and p_{rate} the pacing rate. We note that R_{lat} will be 0 if the increased average latency is smaller than $(1+\beta)d_0$, thus small queuing size will not be penalized. Moreover, we add the pacing rate as a multiplier to penalize sending rate increments in a link that experiences high latency inflation. Thus, R_{lat} can be regarded as "the total increased latency of all sending packets" in an MTP.

We measure both the fairness and stability metrics in variances of throughput, but along different axes. Specifically, we calculate the stability metric based on the standard deviation of a flow's throughput in the fixed-length (w) history in the state block and the fairness metric on the standard deviation of throughputs of all active flows at the same time. We average the fairness metric along the time axis and the stability across flows to avoid the transient effect. Specifically, for fairness, instead of using the instantaneous throughput, we use flows' average throughputs in the last w MTPs to calculate the standard deviation; for stability, we average the standard deviations across flows to obtain a smoother metric. Let $thr_{i,t}$ denote the throughput of the i-th flow in t-th MTP. Note that these metrics are all normalized to guarantee similar rewards in various network conditions. The fairness and stability metrics are defined as follows:

$$R_{fair} = \sqrt{\frac{\sum_i (avg\_thr_i - \frac{1}{n}\sum_i avg\_thr_i)^2}{n(\sum_i avg\_thr_i)^2}}$$
$$R_{stab} = \frac{1}{n}\sum_i \sqrt{\frac{\sum_{j=0}^{w-1}(thr_{i,t-j} - avg\_thr_i)^2}{w * avg\_thr_i^2}} \quad (6)$$

where avg_thr_i is the average throughput of i-th flow in the last w MTPs:

$$avg\_thr_i = \frac{1}{w}\sum\nolimits_{j=0}^{w-1} thr\_c_{i,t-j} \quad (7)$$

It is plain that when R_{fair} and R_{stab} are 0s, the flows in the network achieve the optimal fair and stability equilibriums. Putting the metrics together, we give the reward function as follows:

$$R = c_0 * R_{thr} - c_1 * R_{lat} - c_2 * R_{loss} - c_3 * R_{fair} - c_4 * R_{stab} \quad (8)$$

We bound the reward and scale it to the range $(-0.1, 0.1)$ for each MTP. With the dedicated reward function, the RL training algorithm rewards high throughput, good fairness, and stability while penalizing high latency and loss. We can adjust the coefficients c_0, c_1, c_2, c_3, c_4 to achieve various tradeoffs between these subgoals. In general, we have tuned these coefficients to make sure that all reward terms have similar value ranges to balance their importances, and conducted an extensive search to identify a set of hyperparameter combination that performs robustly across diverse network conditions. The reward function serves as the general goal of the CC scheme under various network environments. By maintaining a consistent reward function, we ensure the RL mechanism can adaptively align the current policy with the performance objective across different scenarios. We give the hyperparameter values in the technical report [22]. It is noted that we do not need to explicitly consider responsiveness in R, because RL agents are trained to maximize the accumulative reward in a horizon of future MTPs, which means that they naturally tend to reach the optimal convergence with maximum rewards as fast as possible.

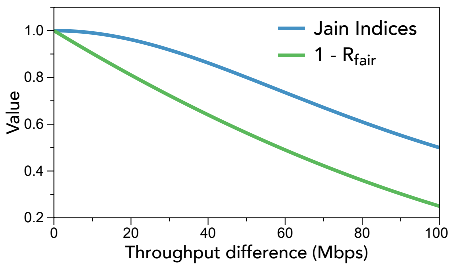


Figure 4. Jain index saturates when throughput difference of two flows approaches zero.

Why not represent fairness metric with Jain index in training? While Jain index [17] is a common metric to evaluate fairness (§5.1), we find that it easily saturates when participant flows are sharing close throughput. We illustrate this phenomenon with a toy example, where there are two competing flows in one bottleneck with 100Mbps capacity and the capacity will be fully utilized by these two flows. The corresponding values across different throughput differences of these two flows are calculated in Figure 4. It shows that the Jain index value is hard to differentiate two throughput with a small difference. For example, when the throughput gap of these two flows increases from 0Mbps to 20Mbps, the Jain index decreases only 0.038, comparing to 0.19 in ASTRAEA's reward[2]. Therefore, ASTRAEA's reward can preserve sensitivity for fairness when participant flows are approaching the equilibrium point, enabling ASTRAEA to keep refining its policy towards the optimal fairness.

3.4 Multi-Agent RL Training

In our multi-flow scenario, we first model the CC problem as a cooperative multi-agent reinforcement learning problem, where each agent represents a flow, and all flow agents are trained together towards a global objective. Then, we adopt a variant of the multi-agent deep deterministic policy gradient utilizing global information to solve the large variance in the multi-agent interaction environment.

Modeling Formally, we formulate CC as a multi-agent extension of the Markov Decision Process called partially observable Markov games [24] with the following specific characteristics: i) it consists of multiple agents/flows interacting with the network environment and each other asynchronously; ii) the flow agents share the same policy and reward objective.

[2] R_{fair} equals zero indicates optimal fairness. Therefore we plot $1 - R_{fair}$ for easy understanding.

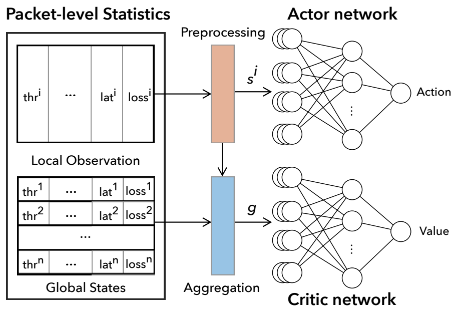


Figure 5. The model architecture in Astraea's training.

At each time step t for the flow i, the global state of the link is defined as the set of local states $(s_t^i, s_t^1, ...s_t^i, s_t^{i+1}, ...s_t^n) \in \mathcal{S}^n$ of all active flow agents as well as other useful global information. Based on the local state $s_t^i \in \mathcal{S}$, the flow agent takes action $a_t^i \in \mathcal{A}$ based on the shared policy $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ denotes the probability distribution over the action space $\mathcal{A}$. After executing the agent's action a_t^i, the state of the network changes according to a state transition function $\mathcal{T} : \mathcal{S}^n \times \mathcal{A} \rightarrow \mathcal{S}^n$ based on the network characteristics. During the state transition, the flow agent obtains a global reward r_t from the environment based on the shared reward function $\mathcal{R} : \mathcal{S}^n \times \mathcal{A} \rightarrow \mathbb{R}$. The objective of all agents is to maximize the cumulative expected return $\mathcal{J} = \mathbb{E}_{(s^1...s^n, a^1...a^n)\sim p^\pi}(\sum_{t=0}^{T} \gamma^t r_t)$ with finite time horizon T, where p^π is the trajectory distribution when all flow agents follow policy π, and γ is a discount factor.

Multi-agent actor-critic training. Training in the multi-flow scenario suffers from large variance when the number of concurrent flows is large, because the reward concerning convergence depends on the interaction between all flows. To train the flow agents, Astraea leverages a customized DRL training algorithm for multi-flow setting inspired by the multi-agent deep deterministic policy gradient (MADDPG) algorithm [25]. By observing the empirical reward return from sampled trajectories, Astraea calculates the gradient of the reward objective and updates the policy of agents. The model architecture is shown in Figure 5. It consists of an actor parameterizing the agent policy π_θ with θ, and a critic for value estimation parameterized with ω. In essence, the critic network functions as a reward predictor and guide, helping the actor network navigate the action space more effectively to maximize the cumulative rewards. Given the local state s^i of a flow agent i, the actor deterministically outputs the action value $\pi_{\theta_i} : \mathcal{S} \rightarrow \mathcal{A}$. Besides, the critic is fed with not only the agent's local state s^i and action a^i, but also the global state g aggregated from the local observations of all active flows in the state block. It approximates the action-value function $Q^{\pi_\theta}(g, s, a) = \mathbb{E}_{s^i, a^i \sim p^{\pi_\theta}}[\sum_{t=0}^{T} \gamma^t r_t | g, a, s]$, which is the expected future reward return from the time step when a flow agent executes action a at the network state g. The technique of using extra global information to guide training in centralized learning paradigm has been employed in many previous multi-agent RL works [15, 25, 31, 32]. With sufficient state features, the critic provides a more precise prediction of the expected reward for the actor, which, fed with only the local link information, learns the best policy to obtain the maximum expected cumulative reward in a more stable way. During the training, we train the critic to precisely predict future reward based on the current state and action, and train the actor to output the action at that state which leads to the maximum collected reward implicated by the critic. Both functions are approximated with deep neural networks, and the model parameters are shared across all flow agents, as they are homogeneous in the environment. The details of our training algorithm are as follows. To update the actor, Astraea calculates the gradient of the objective function according to the deterministic policy gradient theorem [23] as follows:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s,a\sim p^{\pi_\theta}}\left[\nabla_\theta \log \boldsymbol{\pi}_\theta(a \mid s)\, Q^{\pi_\theta}(g, s, a)\right], \quad (9)$$

where $Q^{\pi_\theta}(g, s, a)$ is estimated by the critic network, parameterized as $Q_\omega^{\pi_\theta}(g, s, a)$. By updating the actor parameters with the gradient $\nabla_\theta \mathcal{J}(\theta)$, the flow agents update their policy towards the direction to increase the obtained reward. The critic network is updated following the standard temporal difference method [35] minimizing the objective function defined as follows:

$$\mathcal{L}(\omega) = \mathbb{E}_{s,a,r,s'}\left[\left(Q_\omega^{\pi_\theta}(g, s, a) - r + \gamma Q_\omega^{\pi_\theta}(g', s', a')\Big|_{a'=\boldsymbol{\pi}_\theta(s')}\right)^2\right], \quad (10)$$

where a', g', s' denote the action, the global and local state in the following time step. It measures the relative differences between the Q values for different states and actions. By minimizing $\mathcal{L}(\omega)$ with gradient descent, the critic network is able to provide a good value estimation of $Q_\omega^{\pi_\theta}(g, s, a)$ for the actor to learn towards the correct direction.

In practice, the actor and critic networks are updated based on the sampled experiences. In the training process, the state block collects and preprocesses the local observations from each agent and compiles a global state, denoted as s^i and g. Our training algorithm receives tuples $(g, s_i, a_i, g', s_i', r)$ from the state block, calculates the gradients of $\mathcal{J}(\theta)$ and $\mathcal{L}(\omega)$, and updates the actor and critic networks. The outline of Astraea's training algorithm is shown in Algorithm 1.

Online learning: Due to the centralized training of Astraea, it is non-trivial to retrain our model during the online phase, as the global information may not be available to end hosts. Furthermore, due to the distributed nature of CC, distributed online learning may lead to diversified CC schemes for each end-host, which may cause unpredictable fairness issues between competing flows. Therefore in this paper, we pre-train our CC model offline and distribute the trained model across end hosts. A possibly reasonable continuous

Algorithm 1: Astraea Multi-Agent RL Training

```
Input  : Learning rate $\alpha$ and $\eta$, batch size $B$, episode $T$
Output : Trained model parameters $\theta, \omega$
1 Initialize the actor and critic networks $\pi_\theta, Q_\omega^{\pi_\theta}$ ;
2 for $t \leftarrow 0$ to $T-1$ do
3     Sample $B$ experiences $(g, s_i, a_i, g', s_i', r)$ from the environment;
4     Compute the gradients for the actor:
        $\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s,a\sim p^{\pi_\theta}}\left[\nabla_\theta \log \boldsymbol{\pi}_\theta(a \mid s)\, Q^{\pi_\theta}(g,s,a)\right]$;
5     Compute the gradients for the critic: $\mathcal{L}(w) = \mathbb{E}_{s,a,r,s'}\left[\left(Q_\omega^{\pi_\theta}(g,s,a) - r + \gamma Q_\omega^{\pi_\theta}(g',s',a')\big|_{a'=\boldsymbol{\pi}_\theta(s)}\right)^2\right]$;
6     Update the actor and critic networks:
        $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{J}(\theta), \qquad \omega \leftarrow \omega - \eta \nabla_\omega \mathcal{L}(\omega)$.
```

learning method of Astraea is that the network administrator periodically conducts the retraining with collected network experience and distributes the updated models to end hosts.

4 Implementation

Training environment. We build the training environment of Astraea with Mahimahi [30] and Pantheon-tunnel [2] in Python for compatibility, implementing the functionalities described in §3.2. Mahimahi is used to establish the virtual link for mimicking the network bottleneck, in which every network flow runs in one virtual tunnel built on the top of [2] to guarantee isolation. The communication between network flows and the Controller in the environment is built with UNIX socket for low latency. In particular, we enable asynchronous action fetching and packet statistics collection to mitigate model inference overhead.

Training scheme and hyperparameters. For the multi-agent RL training scheme, we use the same learning rate $\alpha = 0.001$ for both actor and critic networks. Both networks use 3-layer multi-layer perceptrons, with 256, 128 and 64 neurons in each layer. The model is updated for a fixed number of model update step (20 iterations) every time the environment runs for the model update interval (5 seconds). We execute the training on various network conditions so that the learned agent can be generalized to unseen network environments. The environment characteristics are shown in Table 3. More training hyperparameters are given in the technical report [22]. We assign multiple running flows in the link with different RTTs to include heterogeneity. The number of concurrent flows of each training episode is randomly sampled from 2 to 5.

Astraea prototype. Based on the multi-flow training environment and multi-agent RL algorithm, we implement a fully functional Astraea prototype. We implement the sender to leverage Astraea's well-trained model to conduct congestion control, in which fetching information of kernel TCP flows and enforcing actions are built with custom socket options. We have implemented a customized congestion control building block for Linux kernel to bypass regular congestion avoidance from underlying TCP schemes, so that Astraea can fully control the sending process.

Bandwidth	Base RTT	Buffer size factor
[40Mbps-160Mbps]	[10ms-140ms]	[0.1-16]

Table 3. Training environment characteristics.

Astraea inference service. To make Astraea scalable with a large number of concurrent flows, we implement an Astraea inference service which is able to serve multiple Astraea senders. The service is implemented in C++ with TensorFlow C++ APIs for high efficiency. The communication between the inference service and Astraea senders is built with UNIX socket or UDP socket. To reduce the system overhead, the inference service works in a batch mode, i.e., it collects inference requests from multiple senders within a fixed time interval (5ms) as a batch and serves them simultaneously.

5 Evaluation

In this section, with the selected benchmarks in previous work [3, 11], we evaluate the performance of Astraea and demonstrate its advantages over typical TCP schemes (Cubic [14], Vegas [6] and BBR [7]), recent learning-based algorithms (Remy [38], Aurora [18], Vivace [11], Orca [3]) and delay-based scheme Copa [5] through extensive experiments in real-world Internet and emulated environments[3]. Our evaluation centers around several key questions:

- **How does Astraea improve fairness?** We investigate convergence properties of Astraea and other schemes by launching multiple flows in emulated network. Our experiments show that Astraea achieves near-optimal bandwidth sharing among multiple homogeneous flows, achieves 8.4× speed up on convergence time, and delivers more stable throughputs at up to 2.8× compared with Vivace (§5.1.1). We demonstrate that Astraea achieves comparable RTT fairness (§5.1.2) with existing schemes. We then show that Astraea is fair with consistently high Jain Index across various network conditions (§5.1.3) and exhibits near max-min fairness in multi-bottleneck scenarios (§5.1.4). In addition, we seek to understand Astraea's internal working principle to achieve fairness in §5.5. We also conduct sensitivity experiments on the fairness coefficient, and find that Astraea can preserve consistently high Jain indices with a range of coefficient from 0.05 to 0.35 in the technical report [22].
- **How does Astraea improve convergence speed and stability?** Our experiments show that Astraea achieves

[3]The used models and parameters in compared schemes are sourced from their original papers.

8.4× speed up on convergence time, and delivers more stable throughputs at up to 2.8× compared with Vivace (§5.2) In addition, we illustrate the quick responsiveness of Astraea in cellular network environments (§5.2).

- **How does Astraea maintain high performance in emulation and real-world experiments?** First, we show that Astraea exhibits comparable TCP friendliness and is not vulnerable to being inefficient when competing with Cubic flows (§5.3.1). In addition, real-world experiments (§5.3.2) show that, though focusing on convergence properties, Astraea always defines the frontier that shows high throughput with moderate latency. For instance, Astraea achieves 1.4× lower latency inflation than BBR and 3.1× high throughput than Orca. Besides, we present the low CPU overhead of Astraea in §5.4. We provide additional experimental results, including Astraea's performance in varying buffers, satellite networks, and high-speed networks, in the technical report [22].

5.1 Fairness

5.1.1 Fairness of Multiple Homogeneous Flows. To understand how Astraea responds when flows arrive and leave, we emulate a link with 100 Mbps bandwidth with 30ms RTT and 1 BDP buffer. We start 3 flows at the interval of 40s; each flow runs for 120s so that flows co-exist with adequate time length. We omit the results of Aurora in this part since we have shown its unfairness in §2.

We first report the convergence process of each scheme in Figure 6. In general, all TCPs can quickly respond to flow arrivals and departures. Cubic and BBR achieve high link utilization while introducing significant rate oscillations. Also, the delay-based scheme, Copa, can quickly respond to flow events, yet demonstrates significant instability, which might result from the erroneous switches to the competitive mode for a few RTTs [5]. It is interesting to observe that Vivace exhibits relatively slow responsiveness compared with all other schemes, as it needs to perform probing steps before deciding the direction to change rate. Orca mitigates instability compared to its default underlying scheme Cubic. However, its convergence is still far from optimal. The reason is that even if Orca learns to act conservatively to control the rate oscillation issue of its underlying Cubic, it can hardly achieve satisfactory stable fairness through implicit optimization, because the fairness metric is not included in its local reward function. Compared with all these schemes, Astraea exhibits near-optimal fast convergence with efficiency and high stability. The reason is twofold: 1) Astraea explicitly includes convergence metrics in optimization goals and performs efficient training; and 2) it entirely controls the sending behavior, avoiding the variance from underlying TCP schemes.

We then repeat each test 10 times and report the statistical values to describe the fairness property of Astraea. The overall CDF of Jain indices is depicted in Figure 7. The Jain

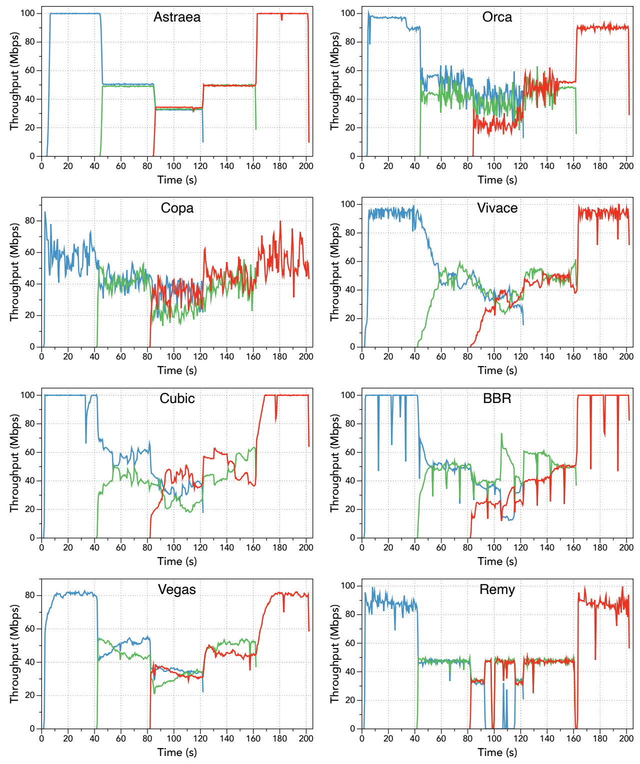


Figure 6. Temporal behavior of convergence of various congestion control schemes.

index is calculated in all the timeslots that have at least two active flows. One key observation is that Astraea achieves almost the full Jain Index all the time, i.e., active Astraea flows can always equally share bandwidth with incoming flows quickly. The distribution of Astraea's Jain Index solidifies the intuition of including convergence properties explicitly in the optimization objective.

5.1.2 RTT Fairness. In addition to studying the competing behaviors of multiple flows that possess the same RTT, we further inspect how Astraea performs when there are multiple concurrent flows with different RTTs. Ideally, flows sharing the same bottleneck should get identical throughputs. To experimentally evaluate the RTT fairness, we set up 5 long-running flows in an emulated link with 100Mbps bandwidth. The base RTT of each flow evenly spaces between 40ms and 200ms. The link has 1 BDP buffer that calculates with 200ms RTT. We repeat each test 10 times. The average obtained throughput is reported in Figure 8.

We observe that Astraea exhibits milder throughput distance to the optimal, showing comparable RTT fairness with Copa and Vivace, and outperforming Aurora, Orca and other TCP schemes. The reason is twofold: 1) We have introduced the RTT heterogeneity in our training setting to explicitly improve Astraea's RTT fairness; and 2) The CWND adjustment in Astraea is RTT independent. As presented in §3.3, the fairness reward signal in Astraea framework does not

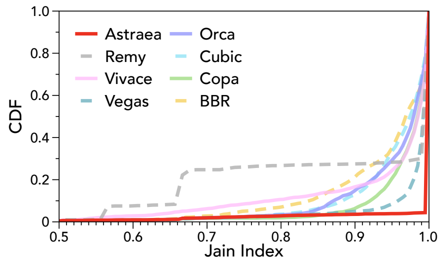


Figure 7. CDF of Jain indices calculated at various timeslots for multiple flow experiments.

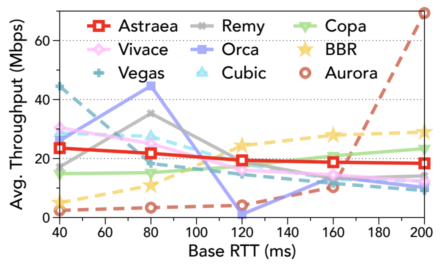


Figure 8. RTT-fairness of various congestion control schemes. 20Mbps throughput means optimal sharing.

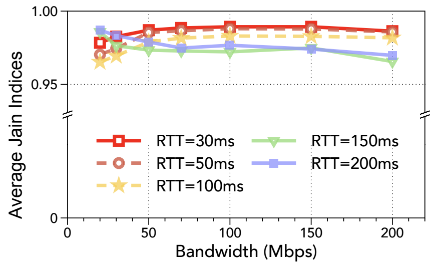


Figure 9. Jain indices in diverse network scenarios.

differentiate RTTs, thus posing the same advantages over them. With this reward logic, multiple heterogeneous flows turn to achieve fair sharing to maximize the incentive of Astraea's objective. However, Astraea still exhibits some unfairness when the RTTs are small. This is because Astraea essentially operates as a time-interval-based algorithm. In small RTT scenarios, Astraea gets faster feedback from the network and therefore responds more swiftly, which leads to its throughput advantage.

5.1.3 Fairness in More Diverse Network Scenarios.

To understand convergence properties of Astraea in more diverse network scenarios, we establish a bottleneck with changing bandwidth and base RTTs, and run multiple Astraea flows inside the bottleneck to observe the degrees of bandwidth fair sharing. The bandwidth and base RTT range from 20Mbps to 200Mbps and 30ms to 200ms respectively (wider than the training range), which we believe can fill the basic range of the Internet. The number of flows of each trial is randomly sampled from 2 to 8. We set the flow starting interval as 20 seconds and tune the running time of each flow accordingly to guarantee the adequate competing similar to §5.1.1.

We report the convergence results by plotting the average Jain indices in Figure 9. Each point in this figure presents the average Jain Index of 10 trials under one specific network configuration. Overall, we find that Astraea achieves high Jain indices across all evaluated network scenarios (higher than 0.95), which demonstrates that Astraea can preserve good fairness across a wide range of network conditions. Furthermore, Astraea provides decent fairness in large RTT scenarios (e.g., 200ms) beyond its training range, which indicates that it can generalize to unseen network conditions.

Besides, we make two additional observations of Astraea's fairness under this wide range of network scenarios: 1) The Jain Index of Astraea degrades under large RTT scenarios. The reason comes from the intrinsic slow feedback from the network, which makes it hard for Astraea to perceive the co-existence of other flows in a timely manner. Therefore, Astraea may perform rate occupation and relinquishing sluggishly and results in slow convergence speed to fairness

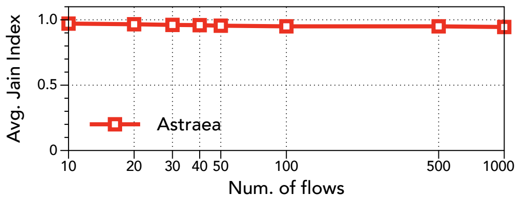


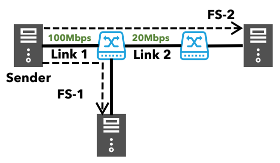
Figure 10. Fairness of varying number of competing flows.

point. Thus, Astraea delivers moderate fairness under network conditions of 150ms and 200ms RTTs. 2) The Astraea fairness performance may degrade in small BDP networks (low bandwidths will small RTTs). For example, there is a performance gap between (50Mbps, 30ms) scenario and (20Mbps, 30ms) conditions. The degradation may be due to the action formulation of Astraea, which will lead to rounding error when the CWND is small, as only integer output is supported.

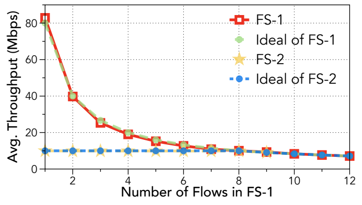
Given that there may be many concurrent flows over the backbone network in Internet [4], we seek to evaluate Astraea's fairness in the scenario of a large number of competing flows. We establish an emulated bottleneck with 600Mbps bandwidth and 20ms RTT and increase the number of competing flows in the bottleneck from 10 to 50 (the maximum number of flows supported by our emulated environment). We also extend this experiment with a larger number of competing flows (up to 1000) in a link using Linux TC qdisc [1]. Figure 10 reports the average Jain Index of 10 trials under different numbers of competing flows. It shows that Astraea preserves a high degree of fairness by exhibiting high Jain indices across all evaluated scenarios, though it is trained with a limited number of flows. We attribute the good generalizability of Astraea's fairness property to our normalization techniques, and §5.5 gives a more illustrative explanation.

5.1.4 Fairness in Multi-bottlenecked Scenarios.

To further show Astraea's fairness properties in the scenario of multiple bottlenecks, we set up a topology in Figure 11a following the same multi-bottleneck setting in [9], in which Flow set 1 (FS-1) and Flow set 2 (FS-2) (representing two sets of flows) share the common Link 1. We set Link 1 and Link 2 to use 100Mbps and 20Mbps bandwidth, respectively,



(a) Topology.



(b) Throughput of FS-1 and FS-2.

Figure 11. Fairness in multi-bottleneck topology.

both of which use the 30ms base RTT with adequate buffers. We establish two flows in FS-2 and vary the flow numbers of FS-1. We start FS-1 and FS-2 simultaneously and present their average throughputs of 10 trials in Figure 11b.

We observe that both the average throughputs of FS-1 and FS-2 closely follow the ideal cases. When the flow number of FS-1 is below 8, its bottleneck link is Link 1 and FS-2's bottleneck is Link 2. As a result, two flows in FS-2 fairly share Link 2's 10Mbps and all flows in FS-1 equally share the remaining 80Mbps in Link 1. When the flow number of FS-1 further increases, Link 1 becomes the common bottleneck for FS-1 and FS-2. In this case, all flows fairly share the 100Mbps bandwidth. The reason why Astraea can deliver good fairness in the multi-bottleneck scenario is that it has been trained to ensure fairness under varying bandwidths and RTTs, which is in fact analogous to the alterations of sending rates in irrelevant flows in multi-bottleneck scenarios. Therefore, the sending rate changes of irrelevant flows will not affect the fairness among flows sharing the bottleneck.

5.2 Convergence Speed and Stability

To demonstrate the convergence process numerically in multi-flow competing scenario, we report the convergence time and stability of experiments in §5.1.1. We calculate the convergence time as the time from flow events (flow arrival or departure) to the time when it reaches a sending rate within ±10% of its ideal fair share. The convergence stability is calculated as the standard deviation of the throughput of the new arrival flow after its convergence. As shown in Figure 12, Astraea achieves the average convergence time of 0.408s, which is comparable with Copa, 3.7× faster than Orca (1.497s), and 8.4× faster than Vivace (3.438s), respectively. In addition, Astraea achieves the best stability (2.124 Mbps) among all evaluated CC schemes, which is 2.6× better than Orca (5.519 Mbps), and 2.8× better than Vivace (6.016 Mbps).

To further understand the responsiveness of Astraea, we evaluate it in a changing network environment. Cellular wireless networks are tricky and challenging for congestion control as their link speed varies drastically in a matter of milliseconds, which requires CC scheme to be agile to bandwidth variances and resilient to random noise. We use Mahimahi to replay the LTE trace provided by [39] and compare Astraea with other schemes in terms of average throughput and normalized latency. The emulated link has 40 ms RTT and a very deep buffer to absorb the traffic. We repeat the test on each CC scheme 10 times.

We first compare the sending rate dynamics of Astraea with Vivace in Figure 13. As shown in the results, Astraea can adjust sending rates swiftly to align with the changes of link capacity, which demonstrates the responsiveness. On the other hand, Vivace cannot capture the changes of link capacity and respond with improper rate adjustment, which therefore causes extreme latency inflation and severe packet losses. The reason is that Vivace's probe-and-decide control logic slows down its reaction to the rapidly changing capacity. It is noted that Astraea is not specially trained for this kind of network environment. More statistical results are presented in the technical report [22].

5.3 High Performance

5.3.1 TCP Friendliness. We study how Astraea behaves when competing with Cubic flows to show its friendliness. We create an emulated link with 100Mbps bandwidth, 30ms RTT and 1 BDP buffer size, and start one evaluated flow with the increasing number of Cubic flows. We repeat each trial 10 times. Figure 14 depicts the average ratio of throughput achieved by the evaluated scheme and the average throughput of other Cubic flows.

We observe that Aurora and BBR yield the worst TCP friendliness (10× to 60× higher throughput than Cubic). Aurora's unfriendliness results from its extremely aggressive behavior. BBR's aggressiveness aligns with results in [11]. Vivace yields apparent throughput disadvantages compared with Cubic, as it essentially operates like a delay-based scheme. On the other hand, Astraea generally achieves acceptable friendliness ratios to Cubic. The reason is that Astraea has learned an adaptive policy: it shows more tolerance to latency inflation when occupying low bandwidth, as shown in Fig. 17, which accounts for its higher throughput than delay-based schemes. However, Astraea still avoids high latency inflation and potential packet loss, therefore it is not as aggressive as BBR and Aurora. The experimental results demonstrate that the benefits of Astraea do not come with aggressive bandwidth grabbing from other flows.

5.3.2 Real-world Experiments. To understand how Astraea performs over wide-area Internet paths with real complex traffic and sophisticated network scheduling policies, we deploy Astraea with other comparisons using Pantheon [42] and evaluate them on the wild Internet. We deploy senders at our residential networks and receivers at AWS nodes. We separate experiments into two categories, depending on the distance between our residential network and AWS nodes: Intra-continent and Inter-continent. We evaluate each CC scheme by running one flow for 60 seconds and repeat each trial 10 times. We summarize the overall average throughput and one-way latency in Figure 15. The ellipse depicts the statistical values of a total of 10 trials.

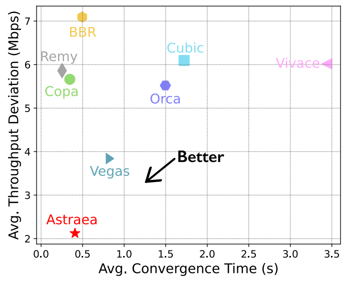


Figure 12. Convergence Time vs Stability.

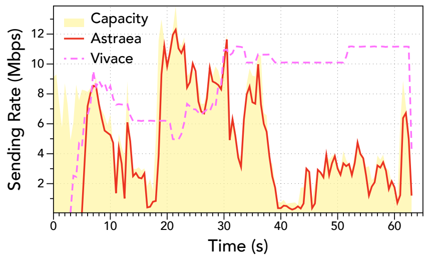


Figure 13. Astraea can adapt to rapidly changing cellular network conditions.

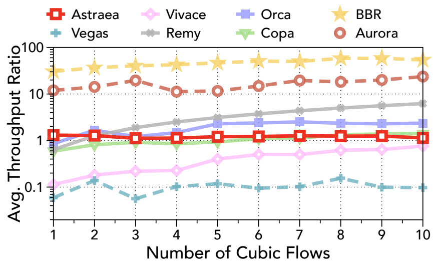
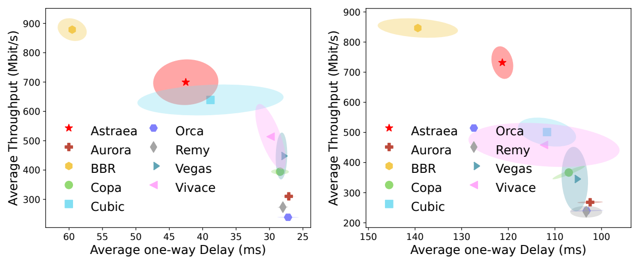


Figure 14. Throughput ratio to CUBIC of various CC schemes. A ratio of 1 indicates optimal friendliness.



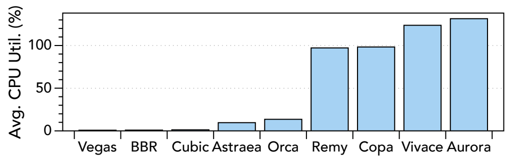
(a) Intra-Continental. **(b)** Inter-Continental.

Figure 15. Overall average throughput vs one-way delay in real-world experiments.

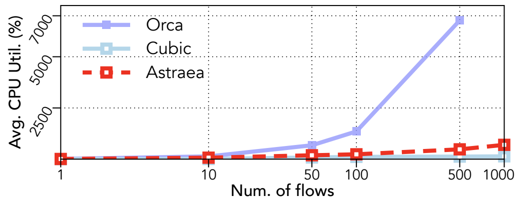
We observe that Astraea performs as the frontiers among these two kinds of experiments. For instance, in the inter-continental scenario, Astraea delivers the average throughput of 731.8Mbps, which outperforms Vivace (458.5) by 1.6× and Orca (236.3) by 3.1×. Among other schemes, BBR achieves the highest throughput while causing explicit latency inflation. Clean-slate machine learning-based schemes, such as Remy, Aurora and Orca do not achieve high utilization, which may be due to the gap between their training environments and the wild Internet. Real-world experimental results demonstrate the generalization ability of Astraea, which means, even trained under the limited range of emulated environments, Astraea exhibits promising performance in the wild Internet. The high performance results from two folds. First, Astraea' fast convergence speed in the single flow scenario ensures that it can quickly adapt to the changing network capacity and thus achieves high utilization. Second, Astraea's reward (§3.3) trades a small amount of latency for higher throughput.

5.4 Overhead

CPU utilization. To understand the computation overhead of Astraea and demonstrate its practicability, we calculate the CPU utilization of each CC algorithm under an emulated link of 100Mbps bandwidth, 30ms RTT and 1 BDP buffer. We set the MI of Astraea to be 20ms, aligning with Orca's default choice. We run each CC for 120s and report the average CPU utilization in Figure 16a. Astraea achieves lower computation overhead than Orca, reducing the overhead by 30%. Astraea's low overhead results from the more efficient implementation of inference service in C++. We also note that the overhead of Astraea can be further optimized by the hierarchical design [37] and in-kernel model execution [43], which we leave as future work.



(a) Average CPU utilization of each CC scheme.



(b) Scalability results of Astraea.

Figure 16. Astraea's CPU overhead and scalability.

Scalability. We further understand Astraea's scalability in Figure 16b. Orca's overhead almost scales linearly with the increasing number of flows, as it requires spawning many inference server instances, which are very resource-inefficient. Our 80-core CPU machine even cannot support 1000 Orca flows. Compared to Orca, the overhead of Astraea does not scale out linearly. This is because Astraea's inference service is able to serve multiple flows, as it executes the inference tasks in the batch manner, which therefore effectively mitigates the overall overhead. We note that Astraea's performance is not impacted by the potential response delay introduced by batch inference (~2ms), as Astraea's control on *cwnd* is not sensitive to fine-grained timing interval. This result shows that Astraea can potentially scale out to a large number of concurrent flows.

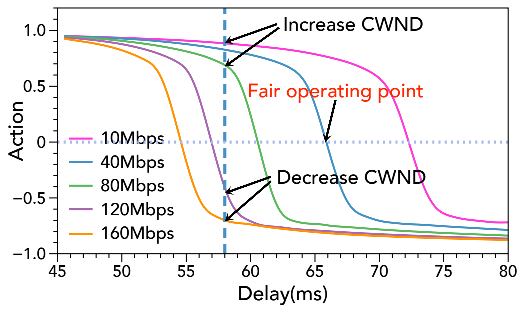


Figure 17. Details of Astraea's action for competing flows.

5.5 Interpreting Astraea's Policy

In this part, we open the black box of DRL model and seek to understand how Astraea achieves fairness across multiple competing flows and how it generalizes the learned policy to more diverse network conditions while preserving convergence properties by visualizing the learned state-action mappings. We fix the max-observed throughput as 200Mbps, the base RTT as 40ms, and plot the state-action mapping of various flows with varying bandwidths and delays in Figure 17.

Our observation is that with the increasing observed delay, all Astraea flows learn to lower the model output actions, transforming from $CWND$ increase to reduction. If there is only one flow in the bottleneck, it will finally reach the equilibrium point of delay where the model action is 0. Moreover, we observe that flows with different bandwidths have different equilibrium points of delay: it increases monotonically with respect to the flow bandwidth. Thus, as all the competing flows on the same bottleneck share the same queuing delay, their divergent sending rate adjustments between flows will cause the bandwidth to transfer from high-throughput flows to low-throughput flows and finally lead to a consensus on the fair operating point, where all flows equally share the link bandwidth with no more sending rate adjustment (action = 0). It can be inferred that as long as the monotonicity stays valid across all feasible bandwidths in the Internet, the fairness property of Astraea under arbitrary numbers of competing flows should be guaranteed, as illustrated in Figure 10.

6 Related Work

We have already discussed Vivace [11] and Aurora [18] in previous sections. In the following, we briefly review classical TCPs and other recently proposed schemes.

Heuristic-based schemes: The core of traditional TCP congestion control research is to figure out what is the congestion signal and how to respond to it. The first category of congestion signal is packet loss. TCP Tahoe, TCP Reno [16], TCP NewReno [12] are the pioneers of loss-based AIMD algorithms. Their followers spent efforts on boosting the speed of linear increasing. (e.g., TCP Cubic [14]). Another well-known congestion signal is delay. TCP Vegas [6] leverages delay inflation as the signal to detect and control congestion. Latter schemes such as TCP Compound [36] utilize a hybrid model that reacts to both delay and loss. Copa [5] is a new delay-based scheme that aims to adjust the congestion window depending on a target rate built from network utility maximization (NUM) [21].

Other learning-based schemes: Orca [3] proposes to couple classical TCP and RL for high practicability. Remy [38] leverages offline optimizations to search the best congestion control logic under a predefined range of network conditions. Indigo [42] utilizes imitation learning to train a deep neural network that encodes mapping from network observations to congestion window adjustments. While Orca's design allows it to integrate with various TCPs, the coupling may inherit the well-known issues of TCPs such as responding to non-congestive loss and TCP unfriendliness. For RL agent, its unawareness of the underlying TCP scheme's behaviors results in difficulties in adapting the learned policies in different networks, where the TCP may behave unpredictably. Besides, the learning method in Orca aims to speed up the training process by parallelly collecting each actor's local experience and optimizes the performance objectives. Astraea differs from it through leveraging the centralized multi-agent RL to study the interactions among flows within the same network, and enabling learning and optimization on the global convergence properties while maintaining high performance.

7 Conclusion

We proposed Astraea, a congestion control algorithm that aims to directly improve fairness, responsiveness, and stability. Astraea leverages multi-agent deep reinforcement learning to build a novel paradigm for improving the convergence properties of congestion control algorithm. Astraea can be naturally coupled with Linux kernel TCP and is hence readily-deployable. Extensive experimental results demonstrate that Astraea shows agile responsiveness and improves fairness and stability significantly in multi-flow scenarios, while maintaining comparable performance with other schemes.

Acknowledgments

We thank the anonymous EuroSys reviewers for their constructive feedback and suggestions. This work is supported in part by Hong Kong RGC TRS T41-603/20R, GRF 16213621, ITF ACCESS, NSFC 62062005, Key-Area Research and Development Program of Guangdong Province (2021B0101400001), and the Turing AI Computing Cloud (TACC) [41]. Kai Chen is the corresponding author.

References

[1] Linux tc. https://man7.org/linux/man-pages/man8/tc.8.html.

[2] Pantheon tunnel. https://github.com/StanfordSNR/pantheon-tunnel. Accessed: 2021-05-30.

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

[4] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *ACM SIGCOMM Computer Communication Review*, 34(4):281–292, 2004.

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

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

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

[8] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 191–205, 2018.

[9] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 239–252. Association for Computing Machinery, 2017.

[10] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

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

[12] Sally Floyd, Tom Henderson, Andrei Gurtov, et al. The newreno modification to tcp's fast recovery algorithm. 1999.

[13] Victor S Frost and Benjamin Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32(3):70–81, 1994.

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

[15] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *ICML*, 2019.

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

[17] Raj Jain, Arjan Durresi, and Gojko Babic. Throughput fairness index: An explanation. In *ATM Forum contribution*, volume 99, 1999.

[18] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning ICML*, pages 3050–3059, 2019.

[19] Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490–2501. IEEE, 2004.

[20] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, 2002.

[21] Frank P Kelly, Aman K Maulloo, and David Kim Hong Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[22] Xudong Liao, Han Tian, Chaoliang Zeng, Xinchen Wan, and Kai Chen. Towards fair and efficient learning-based congestion control. *arXiv preprint arXiv:2403.01798*, 2024.

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[24] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[25] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

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

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

[28] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288, New York, NY, USA, 2019. Association for Computing Machinery.

[29] Gustavo Marfia, Claudio Palazzi, Giovanni Pau, Mario Gerla, MY Sanadidi, and Marco Roccetti. Tcp libra: Exploring rtt-fairness for tcp. In *International Conference on Research in Networking*, pages 1005–1013. Springer, 2007.

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

[31] Tabish Rashid, Mikayel Samvelyan, C. S. D. Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *ArXiv*, abs/1803.11485, 2018.

[32] Tabish Rashid, Mikayel Samvelyan, C. S. D. Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.*, 21:178:1–178:51, 2020.

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

[34] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 479–490, New York, NY, USA, 2014. Association for Computing Machinery.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12. IEEE, 2006.

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

[38] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 123–134, New York, NY, USA, 2013. Association for Computing Machinery.

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

[40] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. Genet: Automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 397–413, 2022.

[41] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. Tacc: A full-stack cloud computing infrastructure for machine learning tasks. *arXiv preprint arXiv:2110.01556*, 2021.

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

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