Towards Fair and Efficient Congestion Control through Multi-Agent Deep Reinforcement Learning

Han Tian[1], Xudong Liao[2], Chaoliang Zeng[3], Xinchen Wan[4], Junxue Zhang[1], Bing Xia[5], Kai Chen[2]

***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$\times$ faster convergence speed and 2.8$\times$ smaller throughput deviation, while achieving comparable or even better performance over prior solutions.

***Index Terms*—Congestion control, Deep reinforcement learning, Transport layer protocols**

I. 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 [1], [2], [3], [4], [5], [6], [7] have been notorious for performance degradation when their assumptions about congestion and packet-level events are violated [8], [9]. To address this problem, a recent evolved thread of research has provided us with a plethora of clean-slate learning-based CC approaches [8], [10], [11], [12], [13], [14], [15], [16]. These learning-based schemes achieve performance goals by defining an objective function consisting of throughput, latency, and loss rate. Offline training [11], [12] or online optimization [10] 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, as shown in §II. The crux, we find, 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 [12]. Moreover, it is difficult to tune the control knobs to achieve a good tradeoff between performance goals and various convergence characteristics, 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 (reinforcement leanring based) CC scheme for the 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. Through joint

Han Tian and Junxue Zhang are with the University of Science and Technology of China (Email: henrytian@ustc.edu.cn, snowzjx@ustc.edu.cn).

Xudong Liao and Kai Chen are with the iSing Lab, Hong Kong University of Science and Technology, Hong Kong SAR, China (Email: xliaoaf@cse.ust.hk, kaichen@cse.ust.hk).

Chaoliang Zeng is with the company BitIntelligence, China (Email: ccllzeng@gmail.com).

Xinchen Wan is with the ByteDance, China (Email: xinchen.wan@bytedance.com).

Bing Xia is with the Zhongyuan University of Technology (Email: xiabing@zut.edu.cn).

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

optimization on global 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 of the network.

Realizing such multi-agent DRL (Deep Reinforcement Learning) 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 (§III-B). 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.

Second, it is not easy to encode convergence properties into an 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 (§III-C). 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 (§III-D), 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 the corresponding environment, and performed efficient distributed training for speedup. We integrate the trained ASTRAEA algorithm with Linux kernel TCP (§IV). Extensive experiments (§V) 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.
- 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 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.

Algorithm	Fairness	Fast Convergence	Stability
Aurora [11]	✗	✓	✗
Vivace [10]	✓	✗	✗
Orca [12]	✓	✓	✗
ASTRAEA	✓	✓	✓

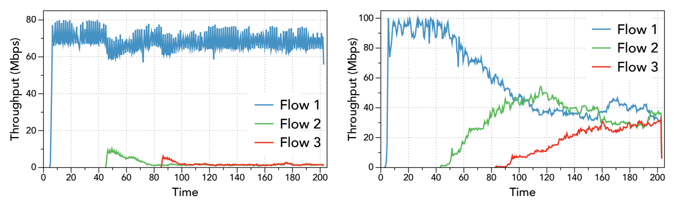
TABLE I: Comparison of learning-based algorithms. None of the existing learning-based algorithms can satisfy all three requirements. Note that Orca is coupled with CUBIC by default, which results in the issue of stability. Besides, as Orca's RL module does not consider optimizing for fairness, its fairness properties are not expected to surpass CUBIC.

II. MOTIVATION

Driven by the tremendous successes achieved by machine learning in computer systems and networks [17], [18], [19], recent proposals seek to leverage learning-based methods to design efficient CC algorithms [10], [11], [12]. 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 I. All of them cannot fulfill fairness, stability, and responsiveness simultaneously.

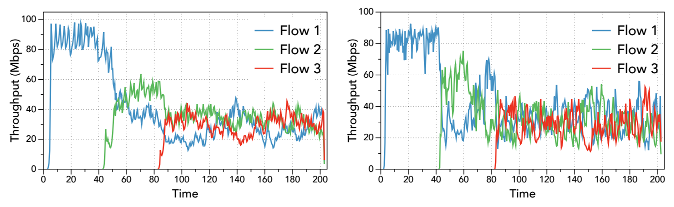
The reason for the failure, we argue, is that their predefined local utility functions are often hard to align with these desiderata, especially *global fairness*. In the following, we first give a brief introduction to two representative learning-based congestion control: Aurora [11] and Vivace [10], and exemplify the objective discrepancy by experimental results.

Aurora is among the first to introduce deep reinforcement learning (DRL) to solve the congestion control, whose core is a carefully designed reward function (Equation 1) that encodes the network utility. In Equation 1, throughput is measured in packets per second, latency in seconds, and loss is the proportion of all packets sent but not acknowledged. 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 hand, Vivace employs a latency-aware utility function (Equation 2), where x_i represents the sending rate (in bytes per second), L_i is the packet loss ratio, and $\frac{d(\mathrm{RTT}_i)}{dT}$ denotes the rate of change in RTT during monitoring interval i. Vivace adjusts the sending rate using



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

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



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

Fig. 2: Enhanced Vivace performs diversely.

online gradient ascent to optimize this utility function. While achieving high performance, they both have different issues related to convergence properties.

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

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

Aurora is unfair. We evaluate Aurora with multiple flows[2] 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 [3], [8]. 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.

[2]Unless otherwise specified, all fairness experiments in this paper assume homogeneous flows, where all competing flows use the same congestion control algorithm.

We attempt to improve Vivace's responsiveness by tuning its *initial conversion factor*, letting Vivace put more rate increment on each probing step ($r_{\text{new}} = r + \theta_0\gamma$) 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 closely. 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 hardly even achieve convergence!

To this end, we argue that the 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.

III. Design

A. Overview

To achieve *explicit* optimization on convergence properties, we design ASTRAEA to directly fulfill the insight derived in §II. 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 (§III-B), RL agents which execute control policy (§III-C), and a Learner which updates the policy using multi-agent RL training algorithm (§III-D).

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 training, all RL agents are initialized with a shared control policy from the centralized Learner. Each flow in the Runtime collects packet-level statistics over fixed time intervals, referred to as Monitoring Time Periods (MTPs) [12], and sends an action request to the Controller with these statistics (i.e., the local state described in §III-C) to obtain a new cwnd. Upon receiving a request from one flow, the Observer relays the request to the

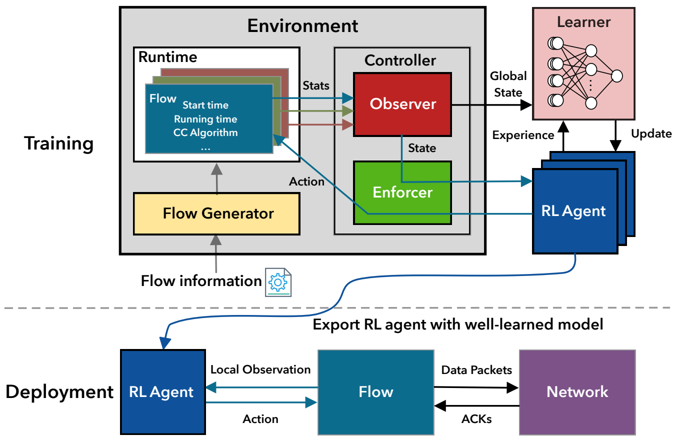


Fig. 3: ASTRAEA Framework.

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 II) 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 the following RL training. Altogether, with global states and 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 to perform congestion control. The 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 in the model input. Therefore, flows from different end-hosts learn to coordinate with each other based on their local observations without further global information about the link characteristics and competing flows, enabling decentralized congestion control.

B. Environment

In general, the ASTRAEA environment is an emulated, monitored, and controlled setting designed for training the policy agent's DNN model. It enables the competition 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 a single flow. Therefore, they can only support direct optimization on performance metrics of throughput, latency, and loss. In contrast, ASTRAEA environment sheds 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 the bottleneck as Poisson arrivals [20] rather than deterministic arrivals. This helps the RL algorithm avoid overfitting to specific, fixed traffic patterns. As a result, ASTRAEA maintains consistent performance across flows with varying arrival times during evaluation. The flexible configuration in our environment allows for the training and evaluation of different metrics. For instance, by using various CC schemes or heterogeneous RTTs, we can study the friendliness or RTT fairness [21] 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 .The 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, the 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 the 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.

C. 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 [11], [12] 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 provided 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 $\text{pkt}_{\text{flight}}$ and the pacing rate as p_{rate}. Our RL agent utilizes the following statistics in an MTP for the local state.

- The throughput ratio, the ratio of the average throughput to the maximum throughput in the flow's history $\frac{\text{thr}}{\text{thr}_{\text{max}}}$.
- 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 $\frac{\text{lat}}{\text{lat}_{\text{min}}}$.
- 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: $\frac{\text{cwnd}}{\text{thr}_{\text{max}} \times \text{lat}_{\text{min}}}$.
- The ratio of loss rate to the maximum throughput $\frac{\text{loss}}{\text{thr}_{\text{max}}}$.
- The ratio of the packets in flight to the current $\frac{\text{pkt}_{\text{flight}}}{\text{cwnd}}$.
- The ratio of the pacing rate to the maximum throughput in the flow's history $\frac{p_{\text{rate}}}{\text{thr}_{\text{max}}}$.

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 III-D. The full definition of global state information is shown in Table II. These values will be used as part of the input for the critic in the training algorithm for better value estimation.

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 II: Global state information in the state block.

State stacking The RL algorithms typically model a task as a vanilla Markov Decision Problem (MDP) [22], where the state transition dynamics depend only upon the present state, meaning that the present state has provided all necessary information for the agent to make the optimal decision. Yet, this assumption falls short in the CC problem. Here, the history data of the network signals and control actions play a crucial role in facilitating the decision-making process for two reasons: i) it allows for the identification of trends in throughput and latency changes caused by other flows, and ii) considering the signal response delay, it requires a history of states and actions to inspect the influence of a flow's control action on the network condition. For instance, a rise in latency might have different implications based on prior actions taken by the agent: If the agent has increased cwnd before, this could suggest that the flow agent itself has saturated the link, making a reduction in the sending rate a wise decision. Otherwise, the latency increase may be due to other flows' aggressive bandwidth use; in this case, there may be no immediate need for the agent to cut back its sending rate.

Therefore, to provide sufficient information for the agent to make the proper decision, the state block in ASTRAEA 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 [11], 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:

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

Based on the updated cwnd, the pacing rate is obtained by dividing the present cwnd by the smoothed RTT of packets $\frac{\text{cwnd}}{\text{sRTT}}$. 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:

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

For the latency metric, [12] 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:

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

where d_0 denotes the base delay and p_{rate} the pacing rate. With the tolerance coefficient β, 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 $\text{thr}_{i,t}$ denote the throughput of the i-th flow in the 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:

$$\begin{aligned} R_{\text{fair}} &= \sqrt{\frac{\sum_i (\text{avg\_thr}_i - \frac{1}{n}\sum_i \text{avg\_thr}_i)^2}{n(\sum_i \text{avg\_thr}_i)^2}} \\ R_{\text{stab}} &= \frac{1}{n}\sum_i \sqrt{\frac{\sum_{j=0}^{w-1} (\text{thr}_{i,t-j} - \text{avg\_thr}_i)^2}{w \times \text{avg\_thr}_i^2}} \end{aligned} \tag{6}$$

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

$$\text{avg\_thr}_i = \frac{1}{w}\sum_{j=0}^{w-1} \text{thr}_{i,t-j} \tag{7}$$

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

$$R = c_0 \times R_{\text{thr}} - c_1 \times R_{\text{lat}} - c_2 \times R_{\text{loss}} - c_3 \times R_{\text{fair}} - c_4 \times R_{\text{stab}} \tag{8}$$

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 trade-offs between these subgoals. In general, we have tuned these coefficients to make sure that all reward terms have similar value ranges to balance their importance and conducted an extensive search to identify a set of hyperparameter combinations that perform 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. It is noted that we do not need to explicitly consider responsiveness in R, because RL agents are trained to maximize the cumulative 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.

Why not represent fairness metric with Jain index in training? While Jain index [23] is a common metric to evaluate fairness (§V-A), 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 makes it hard to differentiate two throughputs with a small difference. For example, when the throughput gap of these two flows increases from 0Mbps to 20Mbps, the Jain index

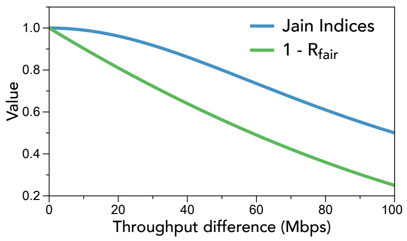


Fig. 4: Jain index saturates when throughput difference is zero.

decreases only 0.038, compared to 0.19 in ASTRAEA's reward[3]. 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.

D. 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. 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 $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, 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 $\mathcal{T} : \mathcal{S}^n \times \mathcal{A} \to \mathcal{S}^n$ 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 $\mathcal{R} : \mathcal{S}^n \times \mathcal{A} \to \mathbb{R}$. 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 a 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

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

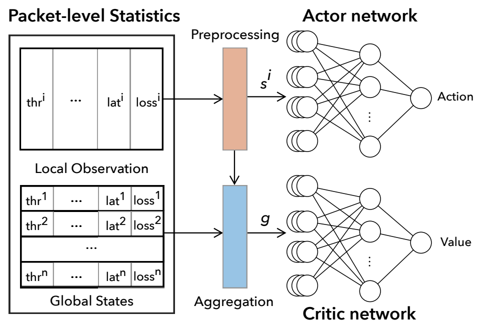


Fig. 5: The model architecture in ASTRAEA's training.

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 $\pi_{\theta_i} : \mathcal{S} \to \mathcal{A}$. 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 a centralized learning paradigm has been employed in many previous multi-agent RL works [25], [26], [27], [28]. 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 that leads to the maximum collected reward implied 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 [29] 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 [30], 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

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: $\mathcal{L}(w) =$
       $\mathbb{E}_{s,a,r,s'}\left[\left(Q_\omega^{\pi_\theta}(g,s,a) - r + \gamma Q_\omega^{\pi_\theta}(g',s',a')\big|_{a'=\boldsymbol{\pi}_{\boldsymbol{\theta}}(s)}\right)^2\right]$;
6     Update the actor and critic networks:
       $\theta \leftarrow \theta + \alpha\nabla_\theta \mathcal{J}(\theta)$,    $\omega \leftarrow \omega - \eta\nabla_\omega \mathcal{L}(\omega)$.
```

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.

Training Optimizations. We adopt several optimizations for ASTRAEA's multi-agent training algorithm. First, we adopt the experience replay memory technique [31] to smooth the learning process. The gathered tuples from agents $(g, s_i, a_i, g', s_i', r)$ are stored in a replay buffer, and the learning thread (denoted as learner) samples experiences from the buffer to calculate the gradients and update the networks. Second, to alleviate the function approximation error in neural networks, we introduce optimizations from TD3 [32]. We refer the reader to [32] and our implementation for the details of these optimizations.

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

IV. IMPLEMENTATION

Training environment. We build the training environment of ASTRAEA with Mahimahi [33] and Pantheon-tunnel [34] in Python for compatibility, implementing the functionalities described in §III-B. 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 top of [34] to guarantee isolation. The communication between network flows and the Controller in the environment is built with a UNIX socket for low latency. In particular, we enable asynchronous action fetching and packet statistics collection to mitigate model inference overhead.

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

TABLE III: Training environment characteristics.

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 steps (20 steps) every time the environment runs for the model update interval (5 seconds). The detailed training hyperparameters are shown in Table **??** in Appendix **??**.

For the environment characteristics, 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 III. We assume no non-congestion loss to simplify the training environment set. 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.

Parallel training across flow agents. The multi-agent training environment in ASTRAEA naturally supports parallel training, as multiple agents act and collect statistics simultaneously in the simulated link. In practice, each agent interacts with the simulated link and with each other, and generates experiences in parallel. This asynchronous training framework has been widely used in the reinforcement learning area and can effectively reduce the training time [35]. For a large-scale distributed training, we also support multiple training environment instances, among which the same actor and critic networks are shared. Our evaluation model is trained with 4 instances.

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 the Linux kernel to bypass regular congestion avoidance from underlying TCP schemes, so that ASTRAEA can fully control the sending process.

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 a UNIX socket or a 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. All evaluation experiments were conducted using the default `fq` (Fair Queuing) qdisc in the Linux kernel.

V. Evaluation

In this section, we evaluate the performance of Astraea using benchmarks from prior work [10], [12], comparing it with conventional TCP schemes (Cubic [2], Vegas [4], BBR [3]), learning-based approaches (Remy [36], Aurora [11], Vivace [10], Orca [12]), and delay-sensitive designs (Copa [37]). Our evaluation answers the following key questions:

- **Fairness:** We assess how Astraea ensures fairness across diverse conditions. Astraea achieves near-optimal bandwidth sharing among homogeneous flows, 8.4× faster convergence than Vivace, and up to 2.8× more stable throughput (§V-A1). It exhibits strong RTT fairness (§V-A2), consistently high Jain Index across diverse settings (§V-A3), and near max-min fairness in multi-bottleneck topologies (§V-A4). We further investigate the mechanism that enables this fairness in §V-E. Furthermore, it fairly coexists with background Cubic/BBR flows under realistic deployment conditions (§V-A5). Astraea also maintains satisfactory throughput and low latency for short flows (Appendix **??**). Astraea not only speeds up convergence but also maintains throughput stability across a wide range of conditions. It is responsive even in mobile environments such as cellular networks (§V-B).
- **Performance Across Networks:** Astraea consistently maintains high performance under a variety of environments, including limited-buffer links (§**??**), satellite (§**??**), cellular network (§**??**) and high-speed backbones (§**??**). It achieves strong real-world performance with high throughput and moderate latency (§**??**). Detailed results are presented in Appendix **??**.
- **Friendliness and Overhead:** We demonstrate that Astraea maintains friendliness to Cubic and BBR and does not exhibit inefficiency when competing with conventional flows such as Cubic (§V-C). Besides, we present the low CPU overhead and good scalability of Astraea (§V-D).

A. Fairness

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 the flows co-exist with adequate time length. We omit the results of Aurora in this part since we have shown its unfairness in §II.

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 [37]. 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 the rate. Orca mitigates instability

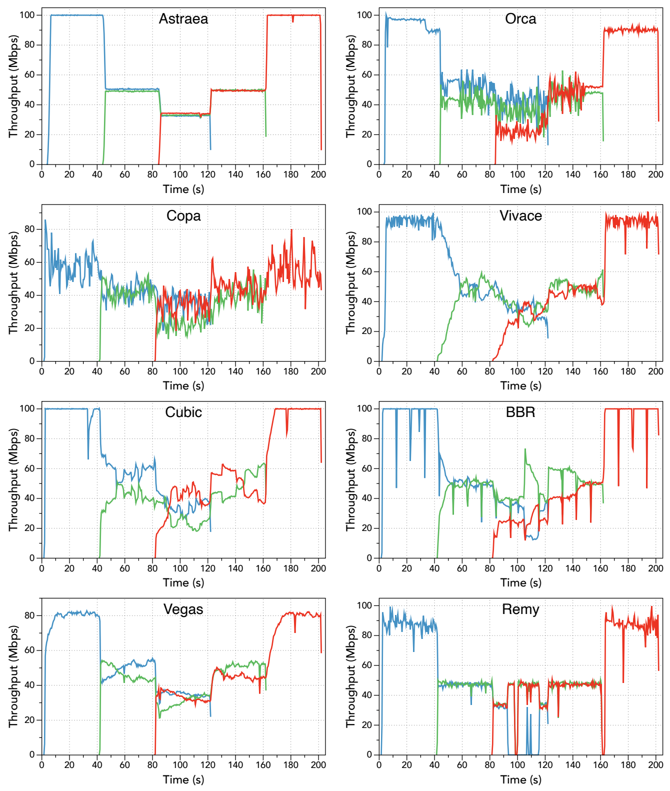


Fig. 6: Temporal behavior convergence of CC schemes.

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 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.

2) RTT Fairness: 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 is evenly spaced 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. The

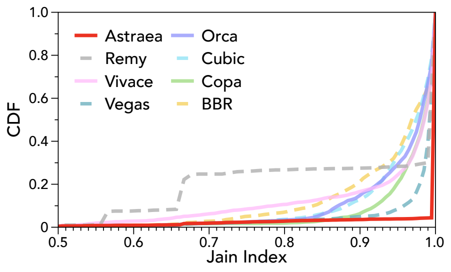


Fig. 7: CDF of Jain indices calculated at various timeslots for multiple flow experiments.

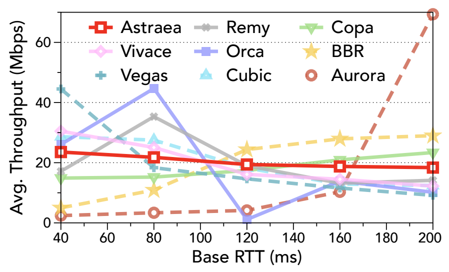


Fig. 8: RTT-fairness of various congestion control schemes. 20Mbps throughput means optimal sharing.

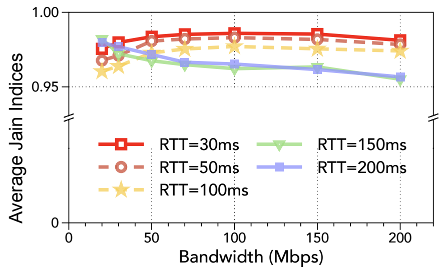


Fig. 9: Jain indices in diverse network scenarios.

variance is less than 5%.

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 §III-C, the fairness reward signal in ASTRAEA framework does not 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.

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. For each network setting above, we iterate through flow numbers from 2 to 8 to ensure comprehensive and consistent coverage. We set the flow starting interval as 20 seconds and tune the running time of each flow accordingly to guarantee adequate competing, similar to §V-A1.

We report the convergence results by plotting the average Jain indices in Figure 9. The variance is less than 5%. Each point in the figure represents the average Jain Index across flow counts 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 resulting in slow convergence speed to the fairness 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.

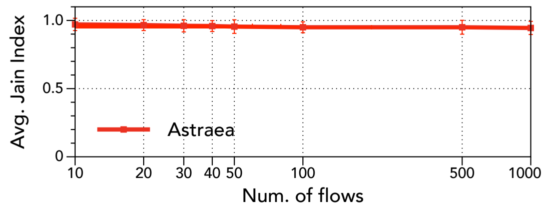
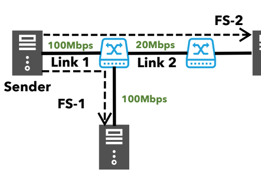


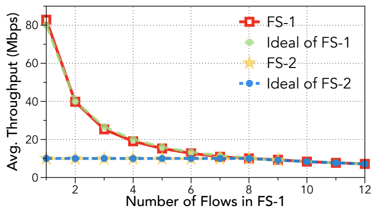
Fig. 10: Average Jain Index of varying number of competing flows with standard deviation.

Given that there may be many concurrent flows over the backbone network in Internet [38], 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 [39]. Figure 10 reports the average Jain Index of 10 trials under different numbers of competing flows. The variance is less than 5%. 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 §V-E gives a more illustrative explanation.

4) Fairness in Multi-bottlenecked Scenarios: To show ASTRAEA's fairness in multiple bottlenecks, we set up a topology in Figure 11a following the same multi-bottleneck setting in [40], in which Flow set 1 (FS-1) and Flow set 2 (FS-

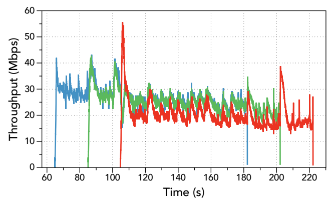


(a) Topology.

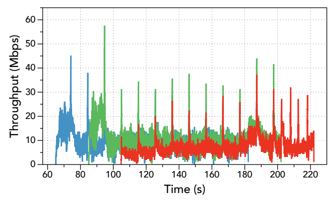


(b) Throughput of FS-1 and FS-2.

Fig. 11: Fairness in multi-bottleneck topology.



(a) Cubic background flows.



(b) BBR background flows.

Fig. 12: Fairness with background flows.

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, 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. The variance is less than 5%.

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) Fairness with Background Flows: To evaluate ASTRAEA's fairness under more realistic conditions with background traffic, we introduce an experiment where non-ASTRAEA flows occupy the bottleneck before ASTRAEA flows join. Specifically, we launch three background flows using traditional congestion control algorithms–either Cubic or BBR–for 5 minutes over a shared 100Mbps link with 30ms RTT and 1 BDP buffer. Subsequently, at 60s, three ASTRAEA flows join with intervals of 20 seconds and run concurrently with the background flows for 2 minutes. The goal is to evaluate whether ASTRAEA can perform well and fairly share the bandwidth in the presence of legacy flows. Figure 12 illustrates the throughput of each flow over time. We observe that once ASTRAEA flows are initiated, they quickly adapt to the existing network dynamics and converge to a fair bandwidth allocation, although the convergence point is lower within BBR background flows due to BBR's aggressiveness.

B. Convergence Speed and Stability

To demonstrate the convergence process numerically in a multi-flow competing scenario, we report the convergence time and stability of experiments in §V-A1. 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

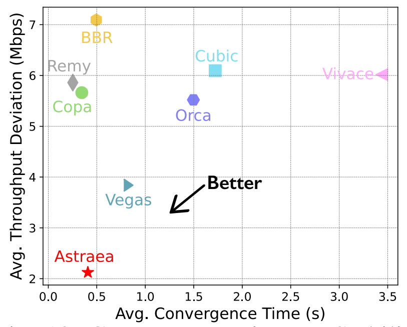


Fig. 13: Convergence Time vs Stability

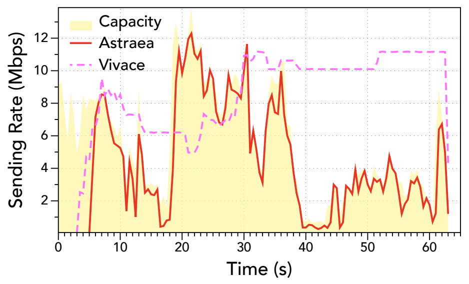


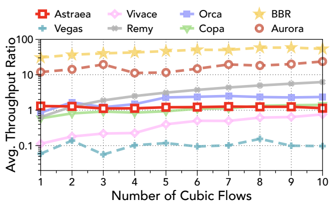
Fig. 14: ASTRAEA adapts to rapidly changing cellular networks.

calculated as the standard deviation of the throughput of the new arrival flow after its convergence. We repeat 10 trials for each method and record the average results.

As shown in Figure 13, 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 [41] 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 compare the sending rate dynamics of ASTRAEA with Vivace in Figure 14. The trace has a granularity of 100 ms, and the plotted results are further smoothed for clarity. 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


(a) Cubic Friendliness.

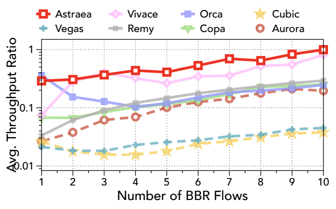

(b) BBR Friendliness.

Fig. 15: Throughput ratio to CUBIC and BBR of various CC schemes. A ratio of 1 indicates optimal friendliness.

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. We observe that ASTRAEA does not fully utilize the link between 16 and 17 seconds, which may be attributed to the lack of training on network environments with ultra-low bandwidth.

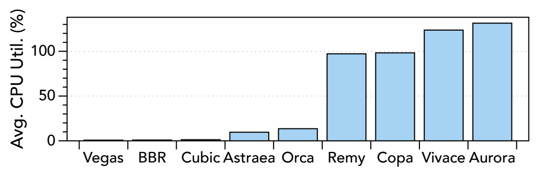
C. TCP Friendliness

We study how ASTRAEA behaves when competing with Cubic/BBR 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 and BBR flows. We repeat each trial 10 times. Figure 15 depicts the average ratio of throughput to the average throughput of Cubic/BBR flows.

For Cubic friendliness, 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 [10]. 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. Regarding BBR friendliness, we observe that ASTRAEA secures good bandwidth and tends to achieve fair allocation as the number of BBR flows increases. These experimental results demonstrate that ASTRAEA's benefits are achieved without aggressively seizing bandwidth from other flows.

D. 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 [15] and in-kernel model execution [42], which we leave as future work.


(a) Average CPU utilization of each CC scheme.

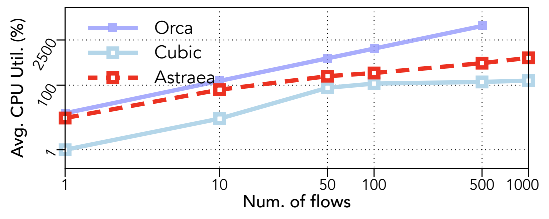

(b) Scalability results of ASTRAEA.

Fig. 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 cannot even support 1000 Orca flows. Compared to Orca, the overhead of ASTRAEA scales sub-linearly with the number of flows. This is because ASTRAEA's inference service is able to serve multiple flows, as it executes the inference tasks in a 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 intervals. This result shows that ASTRAEA can potentially scale out to a large number of concurrent flows.

E. 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,

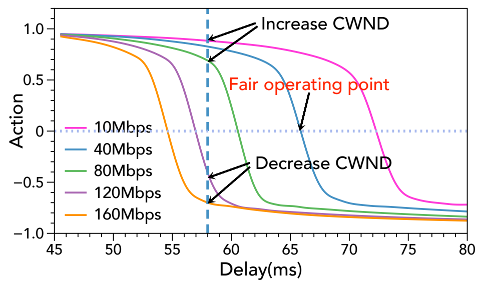


Fig. 17: Details of ASTRAEA's action for competing flows.

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.

VI. Related Work

Heuristic-based schemes: The core of traditional TCP congestion control research is to figure out what the congestion signal is and how to respond to it. The first category of congestion signal is packet loss. TCP Tahoe, TCP Reno [1], TCP NewReno [43] are the pioneers of loss-based AIMD algorithms. Their followers spent efforts on boosting the speed of linear increase. (e.g., TCP Cubic [2]). Another well-known congestion signal is delay. TCP Vegas [4] leverages delay inflation as the signal to detect and control congestion. Later schemes such as TCP Compound [5] utilize a hybrid model that reacts to both delay and loss. Copa [37] is a new delay-based scheme that aims to adjust the congestion window depending on a target rate built from network utility maximization (NUM) [44].

Domain-specific schemes: There are also a lot of efforts providing specially-engineered CC schemes for specific network conditions. Sprout [41] is designed for rapidly changing cellular networks. DCTCP [45] was proposed to replace TCP in high-speed low-latency data center network environments with the help of explicit congestion notification (ECN). Other schemes like DCQCN [46], TIMELY [47], and HPCC [48] are dedicated to RDMA networks.

Other learning-based schemes: Orca [12] proposes to couple classical TCP and RL for high practicability. Remy [36] leverages offline optimizations to search for the best congestion control logic under a predefined range of network conditions. Indigo [49] utilizes imitation learning to train a deep neural network that encodes a 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 agents, their 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.

VII. Discussion

Impact of measurement noise. Measurement noise can influence the model performance by introducing variability or distortion in the input signals used for decision-making. In practical deployments, transport-layer behaviors such as ACK aggregation and delayed ACKs may further affect the accuracy of RTT and throughput measurements. While ASTRAEA is trained across diverse network conditions, it does not currently model these feedback distortion mechanisms explicitly. Explicit modeling of feedback noise into the training environment may allow ASTRAEA to better learn a robust control policy under realistic and imperfect measurement conditions.

Impact of monitoring time period. The choice of MTP can influence the responsiveness, stability, and overhead of learning-based congestion control algorithms. In particular, shorter MTP enables faster reaction to dynamic network changes but increases the frequency of model inference and system overhead, while longer MTP may reduce control responsiveness or result in stale decisions. Recent work, including our prior study SPINE [50], gives a detailed analysis.

VIII. Conclusion

In this paper, we presented ASTRAEA, a multi-agent deep reinforcement learning-based congestion control scheme designed to explicitly optimize for fairness, stability, and fast convergence, in addition to high throughput and low latency. ASTRAEA introduces a cooperative multi-agent training framework and a novel multi-flow simulation environment that enables optimization of global convergence metrics during training. Our extensive evaluation shows that ASTRAEA achieves near-optimal fairness across a wide range of network scenarios. Experimental results demonstrate that ASTRAEA offers a practical and robust solution, bridging the long-standing gap between performance optimization and convergence behavior in learning-based congestion control.

Acknowledgment

This work is supported by the Hong Kong RGC TRS T41-603/20-R, GRF-16213621, NSFC Grant 62402407 and the TACC [51]. Kai Chen is the corresponding author.

References

[1] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.
[2] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Oper. Syst. Rev.*, no. 5, pp. 64–74, 2008.
[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.
[4] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, no. 4.
[5] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM*. IEEE, 2006, pp. 1–12.
[6] C. Jin, D. X. Wei, and S. H. Low, "Fast tcp: motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, vol. 4. IEEE, 2004, pp. 2490–2501.
[7] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, 2002, pp. 89–102.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. USENIX NSDI*, 2015, pp. 395–408.
[9] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan, "An experimental study of the learnability of congestion control," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2014, p. 479–490. [Online]. Available: https://doi.org/10.1145/2619239.2626324
[10] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-learning congestion control," in *Proc. USENIX NSDI*. Renton, WA: USENIX Association, Apr. 2018, pp. 343–356.
[11] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proc. ICML*, 2019, pp. 3050–3059.
[12] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: a pragmatic learning-based congestion control for the internet," in *Proc. ACM SIGCOMM*, 2020, pp. 632–647.
[13] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Owl: congestion control with partially invisible networks via reinforcement learning," in *Proc. IEEE INFOCOM*. IEEE, 2021, pp. 1–10.
[14] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," in *Proc. EuroSys*, 2022, pp. 218–235.
[15] H. Tian, X. Liao, C. Zeng, J. Zhang, and K. Chen, "Spine: an efficient drl-based congestion control with ultra-low overhead," in *Proc. CoNEXT*, 2022, pp. 261–275.
[16] Z. Xia, Y. Zhou, F. Y. Yan, and J. Jiang, "Genet: Automatic curriculum generation for learning adaptation in networking," in *Proc. ACM SIGCOMM*, 2022, p. 397–413. [Online]. Available: https://doi.org/10.1145/3544216.3544243
[17] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM SIGCOMM*, 2018, pp. 191–205.
[18] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.
[19] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2019, p. 270–288.
[20] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Commun. Mag.*, vol. 32, no. 3, pp. 70–81, 1994.
[21] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "Tcp libra: Exploring rtt-fairness for tcp," in *Proc. IFIP Networking*. Springer, 2007, pp. 1005–1013.
[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[23] R. Jain, A. Durresi, and G. Babic, "Throughput fairness index: An explanation," in *ATM Forum contribution*, vol. 99, no. 45, 1999.
[24] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. ICML*. Elsevier, 1994, pp. 157–163.
[25] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv preprint arXiv:1706.02275*, 2017.
[26] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proc. ICML*, 2019.
[27] T. Rashid, M. Samvelyan, C. S. D. Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," *arXiv preprint*, vol. abs/1803.11485, 2018.
[28] ——, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, pp. 178:1–178:51, 2020.
[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[32] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. ICML*. PMLR, 2018, pp. 1587–1596.
[33] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX ATC*, 2015, pp. 417–429.
[34] "Pantheon tunnel," https://github.com/StanfordSNR/pantheon-tunnel, accessed: 2021-05-30.
[35] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*. PMLR, 2016, pp. 1928–1937.
[36] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2013, p. 123–134.
[37] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX NSDI*, 2018, pp. 329–342.
[38] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, 2004.
[39] "Linux tc," https://man7.org/linux/man-pages/man8/tc.8.html.
[40] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*. Association for Computing Machinery, 2017, p. 239–252.
[41] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. USENIX NSDI*, 2013, pp. 459–471.
[42] J. Zhang, C. Zeng, H. Zhang, S. Hu, and K. Chen, "Liteflow: towards high-performance adaptive neural networks for kernel datapath," in *Proc. ACM SIGCOMM*, 2022, pp. 414–427.
[43] S. Floyd, T. Henderson, A. Gurtov *et al.*, "The newreno modification to tcp's fast recovery algorithm," 1999.
[44] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, 1998.
[45] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2010, p. 63–74.
[46] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2015, p. 523–536. [Online]. Available: https://doi.org/10.1145/2785956.2787484
[47] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2015, p. 537–550. [Online]. Available: https://doi.org/10.1145/2785956.2787510
[48] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: High precision congestion control," in *Proc. ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2019, p. 44–58. [Online]. Available: https://doi.org/10.1145/3341302.3342085
[49] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *Proc. USENIX ATC*, 2018.
[50] H. Tian, X. Liao, C. Zeng, D. Sun, J. Zhang, and K. Chen, "Efficient drl-based congestion control with ultra-low overhead," *IEEE/ACM Trans. Netw.*, 2023.
[51] K. Xu, X. Wan, H. Wang, Z. Ren, X. Liao, D. Sun, C. Zeng, and K. Chen, "Tacc: A full-stack cloud computing infrastructure for machine learning tasks," *arXiv preprint arXiv:2110.01556*, 2021.

Han Tian is currently a Associate Researcher at the University of Science and Technology of China (USTC). He received his Ph.D. in the Hong Kong University of Science and Technology, supervised by Prof. Kai Chen and Prof. Qiang Yang. Han's research interests span machine learning and its applications including networking, system and private computing. He has authored and co-authored several referred journals and proceedings including IEEE S&P, NSDI, OSDI, SIGCOMM and EuroSys.

Xudong Liao received the B.Eng. in software engineering from Wuhan University in 2020, and the Ph.D. from the Hong Kong University of Science and Technology in 2025. His research interests span machine learning systems and datacenter networking. He has published several papers in premier conferences and journals, including ACM SIGCOMM, USENIX NSDI, USENIX OSDI, and ACM EuroSys.

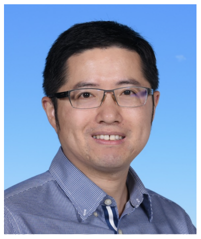
Chaoliang Zeng received the B.S. degree in computer science from University of Science and Technology of China (USTC) in 2018, and the Ph.D. degree from the Department of Computer Science & Engineering, The Hong Kong University of Science & Technology (HKUST) in 2023. His research interests include hardware-accleerated datacenter systems, datacenter networking, and machine learning systems.

Xinchen Wan is a Network Engineer and Researcher at ByteDance. His research interests include AI network, machine learning system, hardware acceleration, and datacenter networking. He received his Ph.D. degree in Computer Science and Engineering from Hong Kong University of Science and Technology (HKUST) in 2025.

Decang Sun is currently a PhD candidate in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. He received his master's degree from Northeastern University in 2020. His research interests include datacenter networking and hardware acceleration.

Junxue Zhang is currently a Professor in the School of Computer Science and Technology at the University of Science and Technology of China. Previously, he served as a Research Assistant Professor at the Hong Kong University of Science and Technology. His research interests include data center networking and machine learning systems. His work has been published in top-tier conferences such as SIGCOMM, NSDI, and CoNEXT.

Bing Xia is an associate professor at Zhongyuan University of Technology. He received his Ph.D. from the Information Engineering University. His research interests include cybersecurity and agent technologies, with a particular focus on innovative applications of intelligent agents in the field of network security.



Kai Chen is the Professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. He received his Ph.D. degree in Computer Science from Northwestern University, Evanston, IL, USA in 2012. His research interests include data center networking, machine learning systems and privacy-preserving computing.