Leveraging Domain Knowledge for Robust Deep Reinforcement Learning in Networking

Ying Zheng*, Haoyu Chen†, Qingyang Duan†, Lixiang Lin†, Yiyang Shao‡, Wei Wang‡, Xin Wang*, Yuedong Xu†
* School of Computer Science and Technology, Fudan University, Shanghai, China
† School of Information Science and Engineering, Fudan University, Shanghai, China
‡Huawei Technologies Co. Ltd., Beijing, China
Email: *{zhengy18, xinw}@fudan.edu.cn, †{haoyuchen17 , duanqy20, lxlin19, ydxu}@fudan.edu.cn,
‡{shaoyiyang, wangwei375}@huawei.com

IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488863

***Abstract*—The past few years has witnessed a surge of interest towards deep reinforcement learning (Deep RL) in computer networks. With extraordinary ability of feature extraction, Deep RL has the potential to re-engineer the fundamental resource allocation problems in networking without relying on pre-programmed models or assumptions about dynamic environments. However, such black-box systems suffer from poor robustness, showing high performance variance and poor tail performance. In this work, we propose a unified Teacher-Student learning framework that harnesses rich domain knowledge to improve robustness. The domain-specific algorithms, less performant but more trustable than Deep RL, play the role of teachers providing advice at critical states; the student neural network is steered to maximize the expected reward as usual and mimic the teacher's advice meanwhile. The Teacher-Student method comprises of three modules where the confidence check module locates wrong decisions and risky decisions, the reward shaping module designs a new updating function to incentive the learning of student network, and the prioritized experience replay module to effectively utilize the advised actions. We further implement our Teacher-Student framework in existing video streaming (Pensieve), load balancing (DeepLB) and TCP congestion control (Aurora). Experimental results manifest that the proposed approach reduces the performance standard deviation of DeepLB by 37%; it improves the 90th, 95th and 99th tail performance of Pensieve by 7.6%, 8.8%, 10.7% respectively; and it accelerates the rate of growth of Aurora by 2x at the initial stage, and achieves a more stable performance in dynamic environments.**

I. Introduction

Inspired by recent successes of deep reinforcement learning (RL) in the machine learning community [1], [2], the conventional computer networking field begins to embrace this new technique to conquer resource allocation problems. Representative applications include job scheduling [3], [4], congestion control [5], adaptive video streaming [6]–[9], and routing [10], [11]. The advantages of Deep RL in networking are twofold. One is the excellent feature extraction ability in dynamic environments. For instance, the job arrival rate and job size are stochastic, the bandwidth available to a user is time-varying. They cannot be accurately predicted by simple machine learning models because their dynamics is further influenced by various network functionalities. Another intriguing merit is that Deep RL offers an end-to-end mechanism of designing networking systems. Simply taking the observable state as input to neural networks, Deep RL is able to explore a very large design space and yield a control policy via a black-box. It lowers down the technical know-how requirement of network operators, thus accelerating the deployment of system prototypes.

Despite of the improved performance, these learning based systems operate in a *black-box* mode which relies on opaque data-driven models. The lack of interpretability makes it difficult to completely trust that these deep models will perform well in real environments, and to debug a problematic decision [12]. For instance, the Deep RL agent in load balancing may place a short job to an overwhelmed server, which is *wrong*; it may leave one or more servers idle, expecting the future arrival of jobs to be short, which is *risk-seeking*. The wrong decisions will degrade the overall performance of Deep RL, and the risky decisions, though beneficial to the average performance, are inclined to causing large performance variance or degrading the tail performance [13]. Therefore, the network operators tend to distrust a decision-making process when the wrong decisions and performance tails have high-stake consequences.

In this paper, we tackle the *robustness* of Deep RL in representative networking systems. The basic idea is to harness the rich domain knowledge to guide the decision-making of Deep RL. Though imperfect and less performant, the domain specific rules can be utilized to identify the problematic actions of Deep RL and provide action advice accordingly. Inspired by [14], we propose a novel and universal *Teacher-Student framework* to improve the robustness of Deep RL based networking systems. The teacher is a small set of simple *white-box* logic rules that are specified by domain specific algorithms or human engineers. The student is an independent Deep RL agent but is infused with teacher's advice. Three key components are developed, including confidence check for locating critical states, reward shaping for injecting teaching data into student network and prioritized experience replay for

This work was supported in part by the Natural Science Foundation of China under Grant 61772139 and Grant 62072117; in part by the Shanghai–Hong Kong Collaborative Project under Grant 18510760900; in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166003; in part by Computer Network and Protocol Laboratory (2012 Laboratory, Huawei). Yuedong Xu is the corresponding author.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.
978-1-6654-0325-2/21/$31.00 ©2021 IEEE

coping with the challenge of limited teaching data.

We implement our Teacher-Student framework in three systems: Pensieve [6] for video streaming, DeepLB for load balancing, and Aurora [5] for TCP congestion control. The teachers are chosen to be the buffer-based algorithm [15], the shortest processing time algorithm and TCP BBR [16], respectively. Comprehensive experiments manifest that our method improves the tail performance (almost) or reduces the variance without sacrificing the average performance, thus making the Deep RL agents more robust. In particular, the real-world evaluation in Pensieve shows that the 90th, 95th and 99th percentile QoE values are improved by 7.6%, 8.8% and 10.7%. When taking the shortest processing time algorithm as the benchmark, our method reduces the standard deviation of DeepLB job processing time by 37%. In NS3 simulations, our method improves the response speed of TCP Aurora (by nearly 2x faster) at the initial climb-up stage, and demonstrate gentle improvements in both the bandwidth utilization and the round-trip time in dynamic network environments.

II. Preliminaries and Related Work

A. Deep Reinforcement Learning

The standard RL problem is formalized as a Markov Decision Process described by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ [17], where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the state transition function, and $p(s'|s,a)$ represents the probability of entering state s' when the agent takes action a at state s. $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function. $\gamma \in [0,1]$ is a discount factor that indicates how much the agent value an immediate reward compared to future rewards. $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ is a policy that describes the probability distribution over all actions at a given state. In Deep RL settings we often use a series of parameters, i.e. neural networks, to approximate policy π so that it can be rewritten as π_θ. More specifically, the agent and the environment interact at discrete time steps, $t = 0, 1, 2, ...$ At time step t, the RL agent observes state $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}$ with its policy π_θ. Following this action, the RL agent receives a reward r_t and the state of environment transits to s_{t+1}. The goal of RL is to find the optimal policy π^*_θ that maximizes the expected cumulative reward denoted by $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

The reinforcement learning algorithms can be divided into two categories, value-based and policy-based algorithms. The policy-based algorithms directly outputs the probability distribution based on which the action is chosen. The policy-based algorithms are suitable for both discrete and continuous actions, e.g. REINFORCE [18] and PPO [19]. The value-based methods output the value function of the state action pair. In greedy cases, the action corresponding to the highest value function is selected. But value-based methods are only suitable for discrete actions such as Q-learning, DQN [2] and many variants [20]. In addition, the Actor-Critic algorithm combines the policy-based and value-based methods. In this paper, we choose to study the PPO and Actor-Critic algorithms because of their good performance and wide adoption in networking.

B. Deep RL in Video Streaming

Adaptive streaming is the predominant form of video delivery service nowadays. A video is subdivided into a sequence of chunks, each of which contains 2$\sim$4 seconds of content and is encoded in multiple bitrates. The higher bitrate means the better video quality. Client-side video players employ bitrate adaptation algorithms to automatically select the chunk bitrates that match the achievable throughput. For instance, the buffer-based algorithm (BBA), the classical rule based policy, uses the buffer length to guide the bitrate selection. When the client-side buffer length is below (above) the minimum (maximum) threshold, the lowest (highest) bitrate is requested. Otherwise, the requested bitrate increases roughly linearly to the buffer occupancy. However, simple logic rules such as BBA need delicate parameter configurations, like the two thresholds mentioned above. They are usually myopic to the current system state, while largely overlooking the dynamics of user throughput in the recent past and the deep interplay between the system state and the bitrate adaptation.

Pensieve [6] tackles this challenge by using Deep RL technique to generate bitrate selection. As shown in Fig. 1, Pensieve's state consists of recent chunk throughput, buffer sizes and actions. Pensieve's action is the bitrate of next video chunk. The goal of Pensieve is to maximize the quality of experience (QoE), a metric consisting of video quality and playback fluency that directly reflecting the perception of users. Pensieve trains its neural network using the Asynchronous Advantage Actor Critic (A3C [21]) algorithm. See [6] for more details.

C. Deep RL in Load Balancing

In a load balancing system, there is one master and multiple working servers. The master appropriately allocates the incoming jobs according to a scheduling policy. Then the server would execute the allocated jobs in a first-in-first-out (FIFO) manner. An effective scheduling algorithm adjusts the scheduling decisions according to the running status of current system with the hope of minimizing average job completion time consisting of waiting time and processing time. The conventional scheduling algorithms require experts to manually configure many parameters. Furthermore, the heuristic logic needs to be re-adjusted upon the changes of environments, e.g. the job arrival patterns. Currently a promising trend is to use deep neural network based algorithms to achieve end-to-end load balancing.

DeepLB. DeepLB is a Deep RL based application that we create based on the load balancer in [22]. The difference between DeepLB and the original version is that the latter does not consider the elapsed time of processing the head of line job in each queue. Omitting this feature will cause the performance degradation in both the Deep RL model and in the heuristic algorithm. Therefore, we redesign the input state and train a new RL agent named DeepLB. Suppose there are k servers in the load balance system. At each decision point t, the RL agent observes state $s_t = (j, q_1, q_2, ..., q_k)$, where j is the size of incoming job, q_k is the sum of k^{th} queue

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

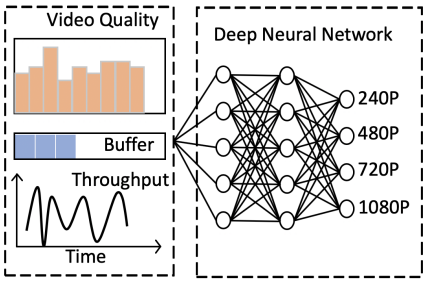

Fig. 1: An example state of Pensieve.

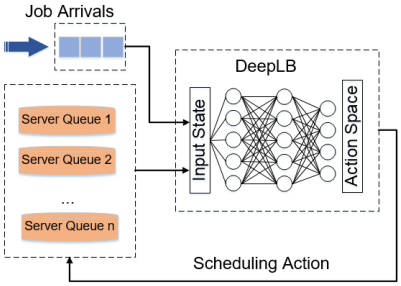

Fig. 2: An example state of DeepLB.

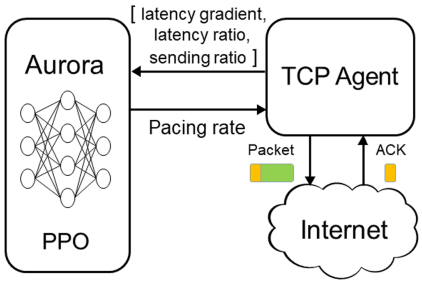

Fig. 3: An example state of Aurora.

size and the remaining size of the current executing job. The RL agent choose an action $a \in \{1, 2, ..., k\}$, indicating the incoming job is allocated to a server, based on this observed state. This policy is often approximated by a neural network due to the huge state space. When a new job arrives at t_i, the reward is calculated by $r_i = -(t_i - t_{i-1}) \times n$, where n represents the number of active jobs in the system during period $[t_{i-1}, t_i]$, t_{i-1} is the timing of last event. Both the arrival and completion of jobs will trigger the calculation of reward. In addition, DeepLB uses Actor-Critic algorithm to train its policy.

D. Deep RL in TCP Congestion Control

TCP congestion control is a natural test field of Deep RL. Owing to the complicated interaction between the senders and the dynamic environment, the traditional algorithms such as TCP CUBIC [23] and TCP BBR [16] may cause the bufferbloat or underutilize the bandwidth. Deep RL enables a TCP sender to learn its transmission rate from experience in which the up-to-date Deep RL congestion controller is TCP Aurora [5].

TCP Aurora uses Proximal Policy Optimization (PPO [19]) as the training algorithm. The state space contains a sequence of network states in the past k monitor intervals. Each network state is a 3-tuple: (i) latency gradient, the derivative of latency with respect to time; (ii) latency ratio, the ratio of mean latency at the current monitor interval to minimum observed mean latency in the history; and (iii) sending ratio, the ratio of transmitted packets to the acknowledged packets. The action space of Aurora is continuous, other than a finite enumerable actions to adjust congestion window size. The action is absorbed as a scaling factor to tune the source sending rate that is implemented through TCP pacing. The reward of a TCP sender is the linear combination of throughput, round-trip delay and packet loss ratio. In terms of their inner workings, BBR and Aurora are very similar, i.e. adjusting the sending rate according to the observed network status. The difference lies in that the former uses the bandwidth delay product to tune the sending interval at the packet granularity, and the latter uses a simple neural network to infer the sending rate at each time interval. Let x_t be the sending rate, a_t be the action at time t, and α be a constant. The control logic of Aurora is

$$x_t = \begin{cases} x_{t-1}(1 + \alpha a_t), & a_t > 0 \\ x_{t-1}/(1 - \alpha a_t), & a_t < 0 \end{cases}. \quad (1)$$

III. Teacher-Student Framework for Robustness

A. Wrong Decisions and Risky Decisions

Deep RL makes decision via a black-box manner. The statistical performance gains often camouflage the deep understanding of their inner-workings. We hereby argue that the Deep RL models can be *wrong* and may take *risky* actions to gain more rewards. The wrong and risky decisions exist in almost all the networking applications in which the load balancer is a superb example for demonstration.

Wrong decision. The incoming short job is assigned to the worker where some long jobs have been waiting in the queue (Fig.4a). The length of the blue blocks represents the processing time of jobs. Since DeepLB is purposed to minimize the average processing time of all the jobs, a proper strategy is to place this short job to the server with a few short jobs.

Risky decision. Suppose that one server is idle and the others have buffered jobs. An incoming long job is not assigned to this idle worker, but is placed at a queue already with long jobs in the front (Fig.4b). DeepLB makes this non work-conserving decision by expecting the future arrivals would be short jobs.

(a) Wrong decision.

(b) Risky decision.

Fig. 4: Wrong decision and risky decision examples in load balancing.

The wrong decision and the risky decision play different roles in the decision making of Deep RL. The former lowers down the average performance, while the latter "entices" the agent to improve the average performance while possibly sacrificing the tails or increasing the variation of performance. Knowing these abnormal decisions requires profound *domain knowledge* that refers to either the direct human perception or the classical rule-based algorithms.

The domain knowledge in networking is far *imperfect*. It might tell a tiny fraction of the wrong decisions using only

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

low-dimensional information (e.g. queue lengths in DeepLB and playback buffer length in Pensieve), and can hardly gauge risky decisions in more complicated scenarios. Meanwhile, the reward of the actions suggested by rule-based algorithms cannot be acquired *immediately*, e.g. the average JCT of DeepLB is measured for a sequence of jobs. In what follows, we aim to put forward a unified Deep RL framework that can harness the imperfect domain knowledge in networking systems.

B. Teacher-Student Learning Framework

The domain knowledge in networking applications, though imperfect, entails us to find wrong or risky decisions. A more robust learning system arises if these domain knowledge can be integrated in the learning systems. Inspired by [14], we adopt a novel Teacher-Student learning framework. The main idea is that RL agents maximize the expected cumulative return, as standard RL does, while minimizing the distance to its teaching data provided by domain knowledge. As one of the primary advantages, our learning framework is universal, orthogonal to both Deep RL models and networking applications.

A neural network defines a conditional probability parameterized by weights θ that maps a certain system state to an action. The Deep RL training updates θ iteratively to produce the optimal actions of training instances. The student is modeled as a MDP process as described in Section II-A and the teacher provides external knowledge to the student network. For the state at which the teacher provides action advice, i.e. the state pertinent to a wrong or risky decision, the RL agent will receive an additional penalty $l(s_t, a_t, a_{tea})$. This new loss quantifies the difference between the student's decision and the teacher's advice. To steer the student network toward the teacher's guidance, we modify the RL objective function:

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \eta\gamma^t r_t(s_t, a_t) + (1-\eta)\mathbf{1}_{\mathcal{H}}(s_t) l(s_t, a_t, a_{tea})] \\
&= \mathbb{E}_{\pi_\theta}[\eta \sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) + (1-\eta) \sum_{t=0}^{\infty} \mathbf{1}_{\mathcal{H}}(s_t) l(s_t, a_t, a_{tea})] \\
&= \eta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)] + (1-\eta)\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \mathbf{1}_{\mathcal{H}}(s_t) l(s_t, a_t, a_{tea})] \\
&= \eta v_{\pi_\theta}(s) + (1-\eta) l_{\pi_\theta}(s) \qquad (2)
\end{aligned}
$$

where $\mathbf{1}_{\mathcal{H}}(s_t)$ is an indicator function with its value equals to 1 when a teacher provides advice at state s_t. η is a constant in $[0, 1]$ that controls the balance between maximizing the expected cumulative reward and imitating the teaching data.

The subsequent challenges are who will play the role of the teacher, and how the teacher's action can be learned by the student network. The teacher is a set of simple *white-box* logic rules that are specified by domain specific algorithms or human engineers. In a word, he should be simple enough and provide exact *action advice* to his neural network based student. In this work, we choose the classical rule-based algorithms as the teachers. The reasons include: (i) they can conveniently provide action advice when "seeing" the states of the student, (ii) though less performant on average, these rule-based algorithms are created with rich domain knowledge and hence are able to provide action advice under certain circumstances.

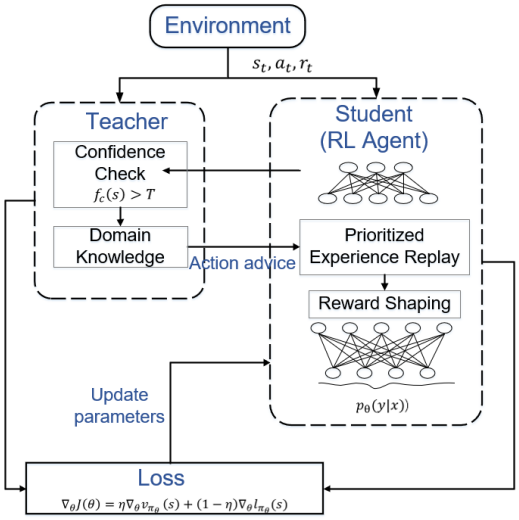

Fig. 5: Teacher-Student learning framework.

The overall Teacher-Student framework is illustrated in Fig. 5. The student is the Deep RL system and the teacher is the domain specific algorithm. The main body of this framework is still the interaction process between the agent and its environment. There are three key modules that help learn teaching data, including confidence check, reward shaping and prioritized experience replay. Among them, the confidence check module can help locate the states of wrong decisions and risky decisions, reward reshaping stimulates the student to learn teacher's advice, and prioritized experience replay achieves effective training when the number of teaching data is not enough. We would give more details about these modules in the following section.

IV. System Design

In this section, we describe three key components of the proposed Teacher-Student framework, including confidence check to automatically locate critical states, reward shaping and prioritized experience replay to effectively incorporate advice.

A. Confidence Check.

1) Challenge 1: When and how to give advice?

The training of Deep RL is an interactive process and each iteration would generate a large amount of trajectory data. Extracting wrong and risky decisions is undoubtedly a problem of finding a needle in a haystack. The prerequisite is to identify the critical states that Deep RL may not function properly.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

2) Solution 1: Inspired by [24], [25], we tackle this challenge by using a process of confidence check, during which the teacher provides advice on a state if the corresponding value of confidence metric is less than a given threshold. The domain knowledge in networking applications could help choose the confidence metric and corresponding threshold value. More specifically, the quality of experience (QoE) metric in video streaming service directly reflects a user's satisfaction of video perception. Since it is a real-time system, we can set the confidence metric as a user's instantaneous reward that captures the QoE of each video chunk. For load balancing, we set the ratio of the incoming job size to the size of server's queue as a confidence metric.

However, as illustrated in [26], the expert knowledge is heuristic and may be inaccurate occasionally. Hence, this knowledge should be scrutinized before being provided to the Deep RL agent. Our intuition is that a valid teaching advice should lead to a higher cumulative reward in the current learning sequence when compared to the student's intended action. We called a critical state s_{tea} if

$$f_c(s_{tea}) < T, R(s_{tea}, a_{tea}) > R(s_{tea}, a_{stu})$$

where $f_c(\cdot)$ is a function for calculating the value of the confidence metric at a given state $s \in \mathcal{S}$. T is the established threshold for corresponding confidence metric. $R(s, a)$ is the cumulative reward from taking action a at state s.

B. Reward Shaping

1) Challenge 2: How can a student network learn advice?

Once the action advice is provided, our goal becomes to embed them into the student Deep RL agent. A naïve approach is that we directly use the action advice provided by the teacher for training. The rationality behind it is to help the agent explore the state and action space better. However, as shown in Fig. 13, our experiments prove that this method is invalid. We need a more efficient method to help the agent learn the teaching data.

2) Solution 2: From Equation (2) we can get

$$\begin{aligned}\bigtriangledown_\theta J(\theta) &= \eta \bigtriangledown_\theta v_{\pi_\theta}(s) + (1-\eta) \bigtriangledown_\theta l_{\pi_\theta}(s) \\ &= \eta \mathbb{E}_{\pi_\theta}[G_t \bigtriangledown_\theta log\pi_\theta(s_t, a_t)] + (1-\eta) \bigtriangledown_\theta l_{\pi_\theta}(s)\end{aligned} \tag{3}$$

where the last equality is because $\bigtriangledown_\theta v_{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[G_t \bigtriangledown_\theta log\pi_\theta(s_t, a_t)]$ and $G_t = q_{\pi_\theta}(s_t, a_t)$ in [18]. In addition, G_t can also be in the form of advantage function $G_t = q_{\pi\theta}(s_t, a_t) - b(s)$ in Actor-Critic, or in the form of $G_t = \frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}(q_{\pi_\theta}(s_t, a_t) - b(s))$ in PPO.

The original Teacher-Student method in [14] sets the loss function to the KL divergence of probability distribution output by the students and that given by the teacher, in which the probability distribution determines the classification result. Similarly, for the RL agent with discrete actions, we can also repress the teacher's action advice in the form of one-hot and calculate KL divergence. The problems are that: (i) this approach is only suitable for discrete action space. Since the output of the agent with continuous action is mean and variance of a normal distribution, it is difficult for a teacher to provide accurate values; (ii) The choice of the parameter η is extremely important and sensitive if we calculate gradients of two parts in objective function separately. This becomes worse when neural networks are over-parameterized.

Reward shaping [27] is an important approach in RL whereby additional rewards are used to guide the learning agent for faster training speed. Inspired by [27], [28], we propose a new updating function for Deep RL that injects networking *domain-specific rules* in the student network by modifying the value of G_t of the corresponding teaching data instead of providing additional rewards. This approach could force the student to learn teaching while not impacting the update of other normal state action pairs. The new updating function of Deep RL agent we use in the Teacher-Student approach is:

$$\begin{aligned}\theta \leftarrow \theta &+ (1 - \mathbf{1}_{\mathcal{H}}(s_t))\delta \bigtriangledown_\theta log\pi_\theta(s_t, a_t) G_t \\ &+ \mathbf{1}_{\mathcal{H}}(s_t)\delta \bigtriangledown_\theta log\pi_\theta(s_t, a_{tea}) F_t\end{aligned} \tag{4}$$

here, the expression $\bigtriangledown_\theta log\pi_\theta(s_t, a_t)$ provides the direction that updates the policy parameter θ so as to improve $\pi_\theta(s_t, a_t)$. Equation (4) takes a step towards this goal and the step size also depends on how large the G_t and F_t are. The shaping function $F : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^+$ is a real-valued function. At a teaching state s_t, $F_t \propto (R' - R)$ where R represents the cumulative reward gained by performing student intended action, and R' is by using the teacher's advised action with $R' \geq R$. Note that in some special systems, we may not be able to obtain R' all the time. For example, obtaining R' requires a step backward of the corresponding environment. This can be achieved in the simulation but not in a real system. In addition, for applications like job scheduling, such queuing systems exist the problem of delayed reward, i.e. a single-step reward signal does not reflect the improvement of the system performance with respect to the teaching action, and hence the accurate R' is unknown. So we recommend to choose this from of F_t in streaming systems, since the instantaneous reward can reflect the true performance of an action to some extent. In other systems, it is chosen based on experience. Nevertheless, the value of F_t is much more robust than the value of η.

C. Prioritized Experience Replay.

1) Challenge 3: How to train the student network with little teaching data effectively?

The amount of teaching data varies considerably across different networking applications and different confidence metrics. We need a more effective method to train the student model with little teaching data available.

2) Solution 3: We employ a prioritized experience replay method in [29] to address this challenge. Experience replay enables RL agents to remember and reuse the history data. Authors in [29] show that prioritizing experience replay data will make the learning more efficient. Since different transitions (i.e. the state and advised action pairs) are not equally useful, we therefore introduce the prioritized experience replay mechanism whose kernel component is the criterion by which

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Leveraging domain knowledge for robust Deep RL

Input: Pre-trained DNN, Domain Specific Rules
Output: Policy π_θ

Initialize teacher replay buffer $\mathcal{H}$, student replay buffer $\mathcal{U}$
for each iteration **do**
  $\Delta\theta \leftarrow 0$
  for each episode **do**
    Obtain $s_t, a_t, f_c(s_t)$
    if $f_c(s_t) < T$ **then**
      Get valid action advice a_{tea} from teacher
      Store transition (s_t, a_{tea}, r_t) in $\mathcal{H}$
    else
      Store transition (s_t, a_t, r_t) in $\mathcal{U}$
    end if
  end for
  Sampling teaching data $\mathcal{H}' \sim P(i) = p_i^\alpha / \sum_i p_i^\alpha$, $p_i = 1/rank(i)$
  for each training data **do**
    Compute weight change with shaping
    $\Delta\theta \leftarrow (1 - \mathbf{1}_{\mathcal{H}'}(s_t))\delta \bigtriangledown_\theta log\pi_\theta(s_t, a_t)G_t + \mathbf{1}_{\mathcal{H}'}(s_t)\delta \bigtriangledown_\theta log\pi_\theta(s_t, a_{tea})F_t$
  end for
  Update weights $\theta \leftarrow \theta + \Delta\theta$
end for

the importance of each transition is measured. The improved cumulative reward value is chosen to measure the importance of different state-action pairs. Note that our approach has a similar function with the TD-error [29] while the latter is not suitable for policy-based training algorithms which are most used by networking applications. In addition, the stochastic method is implemented to ensure a non-zero probability of sampling low priority transitions. Formally, the probability of sampling transition i is given by:

$$P_i = \frac{p_i^\alpha}{\sum_i p_i^\alpha} \tag{5}$$

where α is a positive exponent regarding the weight of priority. Here, $p_i \in (0, 1]$ is the priority of transition i calculated by $p_i = \frac{1}{rank(i)}$ where $rank(i)$ is the rank of transition i according to the ascending ranking of importance.

D. Training

The training process is carried out in two stages, the scratching and teaching phases. In the scratching phase, the student interacts with the environment, continually generating training data and using it to improve strategy iteratively, as the standard training process of Deep RL. When the teaching phase starts, the Algorithm 1 shows the pseudocode of teaching phase, the teacher supervises the student performance and provides advice in the form of an exact action decision when the value of confidence metric $f_c(s_t)$ is less than a threshold. Then the valid teacher advice data would be stored in the teacher replay buffer. The training data also contains two parts, including student experience data that collected through interacting and teaching data that is sampled from teaching replay buffer based on the priorities, which are calculated by how much valuable of an action advice. Note that the teacher only participates in teaching phase, part of training process, for the reasons: (i) reducing teaching cost, (ii) it would fail to explore the action space effectively thus falling into a local optimum if integrating teacher's advice data when the student policy network is not mature. The iterations continue until the "student" converge to a DNN that strikes a delicate balance between the expected cumulative reward optimization and imitating the behavior of a domain-specific logic. This process makes neural network more reliable while consistently performing better than domain-specific algorithms on various environment settings.

V. Performance Evaluation

A. Experimental Setup

We implement the proposed Teacher-Student framework in three representative networking applications using Deep RL: (i) Pensieve [6] for dynamic adaptive streaming over HTTP; (ii) DeepLB for load balancing and (iii) Aurora [5] for TCP congestion control. Unless specified explicitly, the parameter configuration is in line with that of each original work.

1) Video Streaming: Consider a video with a length of 193 seconds that is encoded by H.264 codec at the set of bitrates $\{300, 750, 1200, 1850, 2850, 4300\}$Kbps, and is segmented into 48 chunks. We conduct experiments in both simulated system and real system. In the simulated system, the synthetic traces are generated by using a Markov model in which each state is an average throughput ranging from 0.3 Mbps to 4.3 Mbps at the step size of 0.4 Mbps. The transition probability between different states follows a geometric distribution. Each throughput value is drawn from a Gaussian distribution parameterized with the current average throughput and the uniformly distributed variance between 0.05 and 0.5.

Real System. We implement the Pensieve model and our model in real-world dash.js system. Tensorflow.js is used to store and load the trained model. In our setup, we run a virtual machine on our notebook as the client and a host machine as an IIS web server. The client keeps sending XMLHttpRequests to the server to download the video chunks. The client video player is a Google Chrome browser. We convert the trace files to .bat files and use Dummynet at the client side to control the network bandwidth so that a time varying bandwidth can be emulated with realistic bandwidth traces. The video files in our testing was directly from Pensieve, i.e. the same video durations, bitrates and chunk sizes.

The ***teacher*** we use for Pensieve is the classical buffer-based algorithm (BBA) proposed in [15]. BBA defines a buffer size that is the difference between the downloading and the playback progresses. A larger buffer size means a safer situation against playback interruption. BBA selects the lowest bitrate if the current buffer size is below 5s, selects the highest bitrate if it is above 35s, and controls the request bitrate linearly if it is in between. The confidence metric of

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

the teacher is Pensieve's instantaneous reward that reflects the QoE of the last downloaded video chunk.

2) Load Balance: In our experiment, the arrival of jobs is a Poisson process with the mean inter-arrival time of 70 seconds. We consider the case of two types of jobs, the small jobs and the large jobs. The request processing time of small jobs is uniformed distributed between $20 \sim 50$ units, and that of large jobs is also uniformed distributed but in the range between $200 \sim 300$ units. For each incoming job, it is a small one with probability 0.8 and is a large one with probability 0.2. We use Actor-Critic algorithm for RL training. The actor network is a three-layer fully connected neural network. The number of neurons from the input layer to the output layer is 3, 200, 128, 2 respectively. The output of action network is the probability distribution that allocates an incoming job to the candidate servers. The critic network has the same structure as the action network, except that the number of neuron in the output layer is 1. The output of critic network is an estimate of the value function of the input state.

The confidence metric in load balance is the ratio of the incoming job size to the size of server's queue. The number of confidence values is the same as the number of servers in the system. Each time when the RL agent chooses an action, the confidence values are calculated with respect to each server. The teacher provides action advice when the difference between the confidence value of the selected server and the maximum confidence value among all servers exceeds the given threshold. We choose the shortest processing algorithm (SP) as the ***teacher*** in load balance application. SP calculates the total length of each server queue, including the total size of jobs waiting in the queue and the remaining size of the job currently being processed, and schedules the incoming job to the server corresponding to the shortest queue length.

3) TCP Congestion Control: We implement TCP Aurora in network simulator 3 (NS3) [30] that faithfully complies with the emulation in Pantheon. The TCP Deep RL agent is retrained using OpenAI Gym for NS3. We adopt a well-trained model of TCP Aurora to be the the start point of our retraining. This model is trained using an event-driven simulator because the interaction between the sender and the environment in the real-world is time consuming, much larger than the computation of gradients in the neural network. The simulator produces single link traces with bandwidth between 100 packets per second to 500 packets per second, round-trip propagation delay between 100 ms and 1000 ms, queue size between 2 packets and 2981 packets and loss rate between 0 and 5%. The sender is given an initial pacing rate between 30% and 150% of the link bandwidth. The DNN we used here has 2 hidden layers, with 32 neurons in each layer. The evaluation of the pre-trained and retrained TCP Aurora is carried out in NS3.

We use the classical BBR algorithm as the ***teacher*** whose widsom is to let the sending rate match the bandwidth delay product (BDP). Though BBR is expected to operate at the optimal usage of network bandwidth, it heavily relies on the estimation of BDP. We hereby argue that the BDP estimate

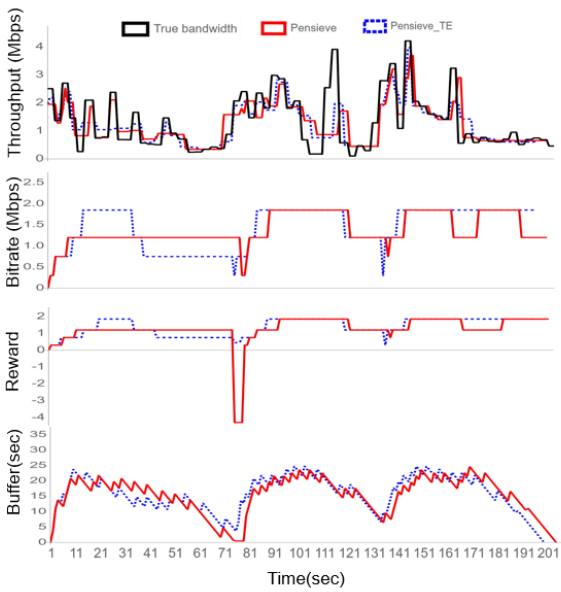

Fig. 6: Comparison of Pensieve and Pensieve-TE.

is not an accurate value to tune the TCP sending rate, but can tell whether the current sending rate is obviously wrong or not. By reviewing the sequence of the states as well as Aurora's actions, we observe that Aurora is sometimes too conservative when the bandwidth utilization is low, and is still aggressive when the network is already congested. We setup two thresholds, α_{min} and α_{max}. If the ratio of the current sending rate over the available bandwidth is below α_{min} or above α_{max}, Aurora will be endowed with an action suggested by BBR. The choice of α_{min} and α_{max} is guided through human experience. In our setting, we set α_{min} to 0.6 and α_{max} to 1.2, and actually they are rather robust to a wide range of configurations.

B. Experimental Results

1) Video Streaming: Since the real video playback is extremely time-consuming (about 10 days for training a converged model), we train the original Pensieve model in the simulated system and test it in the real-world system. A well-trained version of Pensieve is selected and we retrain it by using teaching data provided by its BBA teacher. The instantaneous reward in each single step can serve to indicate the rough quality of the agent's decision. Therefore, we locate the risky decisions based on the QoE value of the downloaded video chunk where the the confidence threshold is -20. Fig. 6 illustrates the throughput, requested bitrate, reward and buffer length of the original Pensieve (red real lines) and the teacher guided Pensieve-TE (blue dash lines) from top to bottom that evolves over time. One can observe that the original Pensieve encounters a rebuffering around 5 seconds at time 73s, and the instantaneous reward decreases significantly. With the actions advised by BBA, the teacher guided Pensieve-TE avoids such a playback interruption.

Although Pensieve and Pensieve-TE have very close average QoE values, the risky decisions made by Pensieve might

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

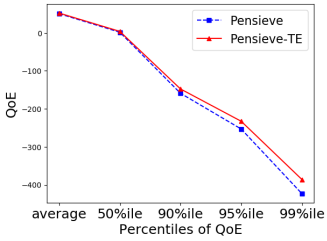

Fig. 7: Percentiles of QoE values.

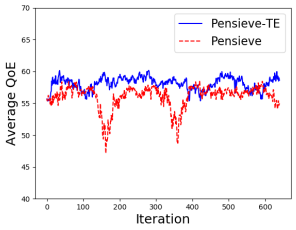

Fig. 8: Average performance during retraining.

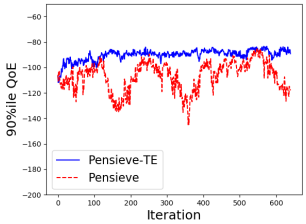

Fig. 9: 90th percentile reward of Pensieve and Pensieve-TE.

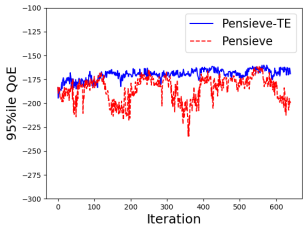

Fig. 10: 95th percentile reward of Pensieve and Pensieve-TE.

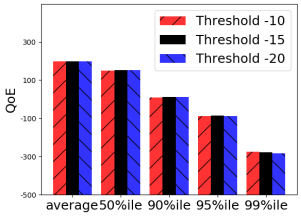

Fig. 11: Using different threshold of confidence value.

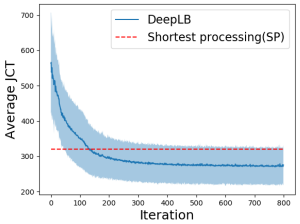

Fig. 12: DeepLB training performance vs training epoches.

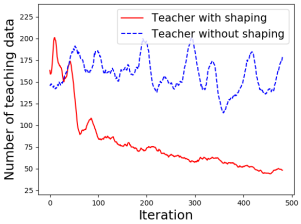

Fig. 13: The number of teaching data w and w/o reward shaping.

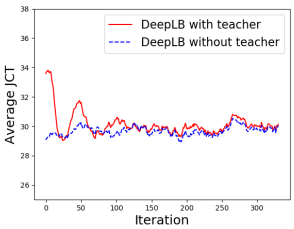

Fig. 14: DeepLB-TE retraining performance.

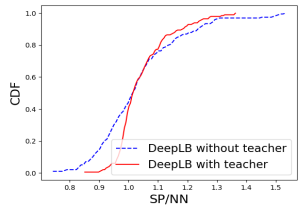

Fig. 15: DeepLB-TE vs Shortest processing algorithm.

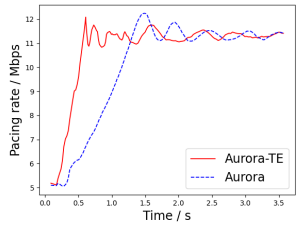

Fig. 16: Number of teaching data of Aurora.

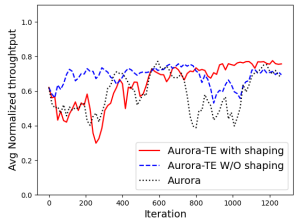

Fig. 17: Normalized TCP throughput.

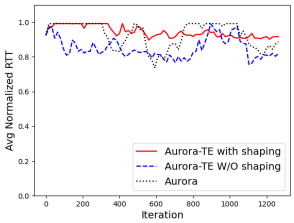

Fig. 18: Normalized TCP RTT.

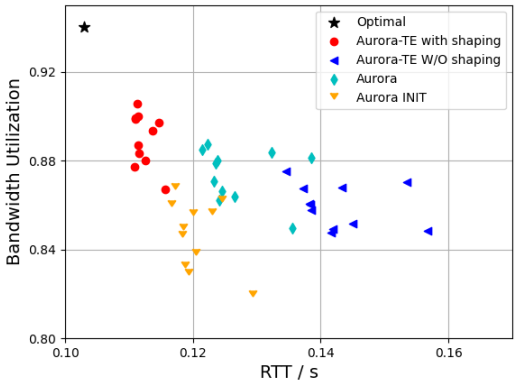

Fig. 19: Comparison of different versions of Aurora.

have caused poor tail performance. Fig. 7 compares their tail QoE values in real system for 300 times, where x-coordinate represents different percentiles of the tails, and y-coordinate displays the average QoE values of these tails. Obviously, Pensieve-TE exhibits a much better tail performance than Pensieve, especially at the 95th and 99th percentiles. Specifically, the proposed method improve the 90%ile, 95%ile, 99%ile performance of QoE by 7.6%, 8.8%, 10.7% respectively. We further conduct a set of experiments to justify the causality that the improved QoE of Pensieve-TE comes from the Teacher-Student framework, other than the insufficient training of the original Pensieve. Fig. 8 shows that Penseive-TE does not sacrifice the average QoE during the retraining epochs, compared with the original Pensieve. Fig. 9 and Fig. 10 demonstrate the average QoE value of the 90th and 95th percentiles during the retraining. We observe that the tail performance of Pensieve-TE gradually improves till convergence, and it is rather stable. In Pensieve-TE, the confidence threshold is a hyperparameter. We next evaluate the sensitivity of Pensieve-TE on the confidence threshold by modifying its value to -10, -15 and -20 respectively. The teachers' advice based on three different confidence thresholds are different, but their outcomes on the tail performance remain to be very close. Hence, we can reasonably claim that the Teacher-Student framework is robust to the setting of the confidence threshold in Pensieve.

2) Load balance: We first train a DeepLB agent from scratch and select a converged model for subsequent enhancement. Fig. 12 shows that as the number of iterations increases, DeepLB outperforms the heuristic algorithm gradually, which manifests the rationality of using Deep RL for load balancing. Note that the shaded area in Fig. 12 describes different test sequences, not the large variance of model performance. We denote by DeepLB-TE the teacher guided model using our

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

Teacher-Student framework. As shown in Fig. 13, x-coordinate is the number of re-training iterations, and y-coordinate is the number of states that the teacher needs to provide advice. The red real curve is for the retrained agent with reward shaping, and the blue dash curve is for that without reward shaping. It is thus highlighted that as the training iteration moves on, the DeepLB-TE with reward shaping relies less and less on the teaching data, but the agent without reward shaping still requires the similar amount of teaching data. This set of experiments implies that the reward shaping is essential to enable the RL agent to learn the advised actions, and to update its own policy network accordingly. Besides, Fig. 14 describes that DeepLB-TE maintains the comparable average job processing time as the original DeepLB.

The Teacher-Student framework improves the robustness of the original DeepLB system. Specifically, we test 200 job sequences and each sequence consists of 70 jobs. We compare the RL agent with the shortest processing time algorithm (SP) that serves as a teacher to provide action advice at critical states. The original DeepLB, though more performant than the rule-based SP in terms of the average job processing time, is found that 42% sequences have worse performance than the rule-based SP. This indicates that the job processing times in DeepLB have a large performance variance. Fig. 15 shows the CDF curves of the performance of DeepLB and DeepLB-TE that are normalized by the performance of SP. With the guidance of the teacher, DeepLB-TE successfully learns the teacher's advice that yield a smaller variance of job processing times. The proposed method reduce the performance variance of DeepLB by 37% when comparing with SP. Hence, DeepLB-TE is believed to be more robust and reliable than DeepLB and SP.

3) TCP Congestion Control: We take a pre-trained TCP Aurora model as the start point and retrain it using our Teacher-Student framework in NS3. Fig. 16 shows the different pacing rate behavior of retrained model Aurora-TE and the initial start point model Aurora. We create a single link with 20 Mbps bandwidth and 40 ms round-trip propagation delay. A UDP background flow with sending rate of 8Mbps is always active. We set the initial pacing rate to 5Mbps for both models. Experiments show that Aurora needs about 1.5 seconds to catch up the optimal pacing rate. But after retraining with intervention of BBR, Aurora-TE only takes 0.7 second, which is nearly 2x faster. This improvement can help the TCP flow utilizes bandwidth more efficiently, especially in the link with dynamically changing bandwidth.

To evaluate whether our approach will harm the average performance of TCP Aurora, we retrain three Aurora models using different retraining methods. After every 16 iterations, for each model we run 55 traces with the same link parameters as they were in retraining setting. Since the bandwidth and latency of every trace are different, we use the normalized throughput and the normalized round trip time (RTT) as defined below to show the performance of models on each trace. The normalized throughput is computed as the throughput divided by the link bandwidth, and the normalized RTT is computed as the fixed round trip propagation delay divided by the RTT. They both should be smaller than 1, and the bigger value means the better performance.

Fig. 17 and Fig. 18 illustrates the average normalized throughput and average normalized RTT of three Aurora models as the training iteration goes on. Aurora-TE with shaping is retrained in our standard Teacher-Student approach. Aurora-TE W/O shaping is also retrained in our Teacher-Student approach but without Reward Shaping. Aurora is just retrained using NS3 traces without teacher's advice. Aurora-TE with shaping achieves a more stable performance compared to the start point model. We also find that Aurora-TE W/O shaping behaves unstable and exhibits a worse RTT performance. It signifies the importance of reward shaping in our Teacher-Student approach.

Finally, to evaluate the scalability and robustness of our retrained Aurora-TE, we consider a dynamic link whose bandwidth varies to a new value randomly chosen between 10 Mbps and 20 Mbps every 5 seconds. The link has a fixed round-trip delay of 100ms. We run 11 times for each model and each run lasts 20 seconds. The comprehensive results are shown in Fig. 19. Aurora INIT represents the start point model or the pre-trained model without any retraining. As expected, these models have different trade-off between RTT and bandwidth utilization. Among them, Aurora-TE with shaping achieves the best performance with 5% higher throughput and 6% lower RTT than Aurora INIT.

VI. Conclusion

In this paper, we shed light on how the domain knowledge in networking is leveraged to improve robustness of Deep RL based learning systems. A Teacher-Student learning framework is proposed in which a domain-specific algorithm serves as the teacher to provide advice to the student neural network. The evaluation results show that: (1) the proposed method is suitable for both the discrete and continuous action space; (2) the proposed method successfully reduce the performance variance, improve tail performance and enable a more stable Deep RL agent while not sacrificing the average performance; (3) reward shaping plays an important role when incorporating the teaching data into student network, i.e. directly using action advice of teacher cannot improve the robustness of student network; (4) the proposed method is robust to different hyperparameter settings. In future work, we plan to extend the extend the learning framework to valued-based Deep RL algorithms.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] 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.

[3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks(HotNets)*. ACM, 2016, pp. 50–56.

[4] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication(SIGCOMM)*. ACM, 2019, pp. 270–288.

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

[6] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication(SIGCOMM)*. ACM, 2017, pp. 197–210.

[7] Y. Guan, Y. Zhang, B. Wang, K. Bian, X. Xiong, and L. Song, "Perm: Neural adaptive video streaming with multi-path transmission," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1103–1112.

[8] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1967–1976.

[9] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 1208–1216.

[10] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks(HotNets)*, 2017, pp. 185–191.

[11] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully-distributed multi-agent deep reinforcement learning," *arXiv preprint arXiv:1905.03494*, 2019.

[12] Y. Zheng, Z. Liu, X. You, Y. Xu, and J. Jiang, "Demystifying deep learning in networking," in *Proceedings of the 2nd Asia-Pacific Workshop on Networking*, 2018, pp. 1–7.

[13] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[14] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing, "Harnessing deep neural networks with logic rules," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 2016, pp. 2410–2420.

[15] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication(SIGCOMM)*, 2014, pp. 187–198.

[16] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Communications of the ACM*, vol. 14, no. 5, pp. 20–53, 2016.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems(NIPS)*, 2000, pp. 1057–1063.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[21] 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 *International conference on machine learning(ICML)*, 2016, pp. 1928–1937.

[22] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *arXiv preprint arXiv:1807.02264*, 2018.

[23] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[24] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning." in *ESANN*, 2008, pp. 143–148.

[25] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[26] K. Efthymiadis and D. Kudenko, "Knowledge revision for reinforcement learning with abstract mdps," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems(AAMAS)*, 2015, pp. 763–770.

[27] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[28] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, D. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," 2017.

[29] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[30] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on September 10,2024 at 10:17:34 UTC from IEEE Xplore. Restrictions apply.