Towards Safe Load Balancing based on Control Barrier Functions and Deep Reinforcement Learning

Lam Dinh, Pham Tran Anh Quang, Jérémie Leguay
Huawei Technologies Ltd., Paris Research Center, France
{lam.dinh, phamt.quang, jeremie.leguay}@huawei.com

***Abstract*—Deep Reinforcement Learning (DRL) algorithms have recently made significant strides in improving network performance. Nonetheless, their practical use is still limited in the absence of safe exploration and safe decision-making. In the context of commercial solutions, reliable and safe-to-operate systems are of paramount importance. Taking this problem into account, we propose a safe learning-based load balancing algorithm for Software Defined-Wide Area Network (SD-WAN), which is empowered by Deep Reinforcement Learning (DRL) combined with a Control Barrier Function (CBF). It safely projects unsafe actions into feasible ones during both training and testing, and it guides learning towards safe policies. We successfully implemented the solution on GPU to accelerate training by approximately 110x times and achieve model updates for on-policy methods within a few seconds, making the solution practical. We show that our approach delivers near-optimal Quality-of-Service (QoS) performance in terms of end-to-end delay while respecting safety requirements related to link capacity constraints. We also demonstrated that on-policy learning based on Proximal Policy Optimization (PPO) performs better than off-policy learning with Deep Deterministic Policy Gradient (DDPG) when both are combined with a CBF for safe load balancing.**

***Index Terms*—Software Defined-Wide Area Network (SD-WAN), Deep Reinforcement Learning (DRL), Control Barrier Function (CBF).**

I. Introduction

Many enterprises are adopting Software Defined-Wide Area Network (SD-WAN) [1] technologies to trade-off between cost-effectiveness and Quality-of-Service (QoS) satisfaction. Relying on a network overlay, this architecture allows businesses to interconnect multiple sites (enterprise branches, headquarter, data centers) without the need to deploy their own physical infrastructure, making it cost effective. A centralized controller maintains a set of policies deployed at access routers which send traffic to their peers over several transport networks (e.g., private lines, broadband internet, 5G). Typically, access routers are responsible for enforcing traffic engineering, for instance load balancing, and queuing policies to meet Service Level Agreement (SLA) requirements in terms of end-to-end Quality-of-Service (QoS), security, etc. At a slow pace, the controller maintains policies, while access devices make real-time decisions for every flow. SD-WAN provides a cost efficient alternative to private lines and drastically improves QoS compared to best effort solutions such as Virtual Private Networks (VPN).

Several solutions have been proposed for load balancing in SD-WAN networks to satisfy SLA requirements. Centralized and distributed path selection mechanisms have been proposed to maximize an utility function [2] or SLA satisfaction using a performance model [3]. To optimize latency and other QoS parameters for a specific set of flows, closed-form performance models, using network calculus or queuing models, can be embedded into routing optimization algorithms. For instance, authors in [4] considered the Kleinrock function [5] to minimize latency. However, accurate analytical models for end-to-end QoS performance metrics can be difficult to derive and integrate into routing optimization algorithms. Indeed, unknown scheduling parameters, behaviors inside the WAN and transport-layer mechanisms are, in practice, too difficult to capture by tractable performance models. The challenge to integrate accurate analytical models makes QoS routing and lod balancing a great opportunity for model-free solutions.

Instead of having a predefined performance model, Reinforcement Learning (RL) agents can interact with the environment and evaluate the outcomes of their actions thanks to a reward function. Deep Reinforcement Learning (DRL) [6] combines Deep Learning (DL) and Reinforcement Learning (RL) principles (e.g., parameterize policies with neural networks). It has been first applied for routing to optimize network utility under the umbrella of *experience-driven networking* [6]. Since, several single-agent and multi-agent DRL solutions have been proposed to tune queues and load balancing policies to satisfy QoS requirements or minimize congestion [7]–[12]. However, even if DRL has demonstrated a tremendous potential for improving SD-WAN performance, most of the literature only focuses on off-policies and their performance once training has converged, without paying attention to safety during both learning and testing. In addition, as network environments often drift, frequent retraining of the full model to adapt to changes can be cumbersome. In DRL, the trial-and-error process is crucial for environment exploration and learning but, at the same time, unsafe actions cannot be deployed as they degrade network quality in production systems. For example, a poor load balancing policy might cause severe congestion and accidentally degrades the overall network performance. Therefore, considering safety during both training phase, in particular for more practical on-policies, and testing is key for the wide adoption of RL algorithms in network systems.

Taking those issues into consideration, this work seeks to complement current DRL-based load balancing solutions with an additional safety shield. Based on the safe learning approach primarily presented in [13], which is designed for critical robotic systems, we propose a safe load balancing solution for SD-WAN. Our solution achieves near-optimal Quality-of-Service (QoS) performance in terms of average end-to-end delay, while safety concerns related to the violation of link capacity constraints are fully considered.

The contributions of our work are the following. First, we describe the target SD-WAN system and formulate a load balancing problem to minimize the average tunnel latency that can efficiently be solved with RL. Then, we design a dedicated Control Barrier Function (CBF) based on local search to deliver safety on top of gradient-based Deep Reinforcement Learning (DRL) algorithms (e.g., off/on policy learning). Finally, we evaluate our DRL-CBF solution in both training and testing phases. We compare our solution to traditional learning algorithms (e.g., DDPG, PPO) where safety is only handled in the reward function, without any strict guarantees. We show that our solution can minimize latency while providing full safety guarantees. In a controlled environment, we also demonstrate that the QoS obtained is very close to the optimal solution derived from a non linear integer model solved with the SCIP [14] solver. In addition, in terms of execution time, we implemented DRL-CBF algorithms on GPU and managed to accelerate training by approximately 110x times and achieve model updates for on-policy methods within a few seconds, making the full solution practical.

The paper is organized as follows. Related work is discussed in Section II. Section III presents the system model and formulates the load balancing problem. Section IV describes the proposed solution, whereas Section V provides numerical results, demonstrating our proposed algorithm performance. Finally, Section VI concludes the paper.

II. Related work

Deep Reinforcement Learning (DRL) algorithms for SD-WAN controllers has been proven to improve overall network performance [7]–[12]. For instance, Troia et al. [11] have shown a target QoS can be achieved through the proper design of the reward function. Similar results have been achieved using multiple agents [9]. However, as mentioned before, most of the literature only focuses on 1) off-policies and 2) network performance without paying attention to safety.

Indeed, RL-based load balancing systems may violate capacity constraints both in training and testing phases. When globally minimizing the end-to-end delay, unsafe actions creating congestion and violating capacity may be taken to get rid of portions of the traffic and improve the reward. To deal with such abnormal behaviors, the LearnQueue [15] reward has been introduced to minimize the end-to-end delay while penalizing traffic rejections. However, it requires to weight properly the two objectives, which can be tedious in practice since it heavily depends on the environment. To address these limitations and avoid manual parameter tuning, the first work to systematically optimize QoS under safety constraints for load balancing has been presented by Kamri et al. [16]. This work applies the Reward Constrained Policy Optimization (RCPO) algorithm [17] where the reward integrates traffic rejection as a constraint using Lagrangian relaxation. During training, the algorithm finds the optimal Lagrangian multiplier. Following this work, Zhang et al. [18] investigate a path planning problem based on constrained policy iteration to improve the performance of multiple path selection. A safe load balancing strategy for ultra-dense network is also discussed by Huang et al. [19]. In this work, they proposed a proactive load balancing algorithm on top of Constrained Policy Optimization (CPO) [20], which has been proven to guarantee optimal policy under (safety) constraints.

Although several papers have presented DRL algorithms for load balancing [11], [15] with some constraints [16], [19]), they 1) all provide *soft* guarantees during exploration and 2) mostly ensure safety during exploitation. To mitigate these issues, we propose to employ a Control Barrier Function (CBF) [21] on top of current DRL algorithms for (safe) load balancing optimization with *hard* guarantees.

III. System model

Figure 1 presents a typical SD-WAN use case where the headquarter and 3 branches of an enterprise are interconnected either via an Internet connection and a Multi-Protocol Label Switching (MPLS) private line. Traffic is issued by applications at both headquarter and branches. 6 OD (Origin-Destination) flows, also called *tunnels*, are considered, one per headquarter and branch pair in each direction. Each tunnel has two paths for Internet and MPLS. A Load Balancer (LB) agent at each Access Router (AR) splits the traffic according to the policy received by the centralized controller.

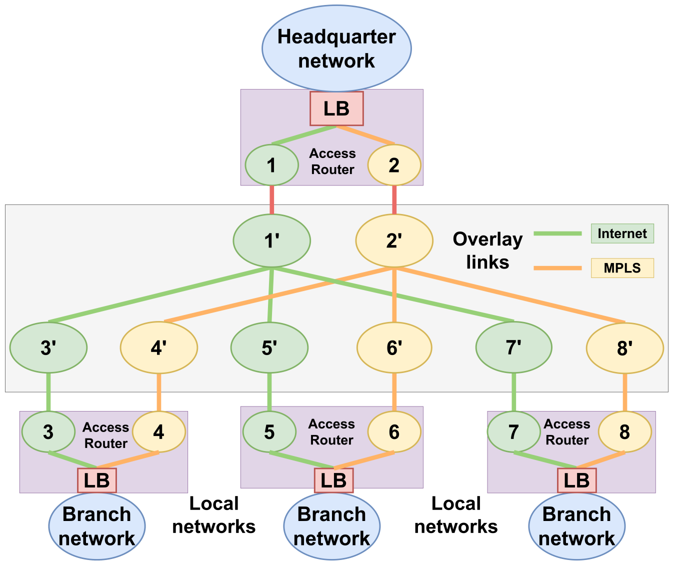


Fig. 1: SD-WAN network with an headquarter and 3 branches.

This overlay network can be modeled as a graph where core links represent overlay links and edge links correspond to WAN ports at AR routers. As depicted in Figure 1, links $N' - N$ at ports N are used to model the egress / ingress capacity provisioned at each transport network. In practice, WAN ports can receive traffic from multiple sites, potentially of higher capacity, and they might be congested in high load conditions. Let formally consider a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. Each tunnel k in a set of tunnels K can use a set of *candidate paths* denoted as $\mathbb{P}_k$ (e.g., Internet and MPLS) to load balance traffic. Each edge e carries an instantaneous load l_e and has a capacity c_e. Let denote T^k the traffic demand of tunnel k at time t. Each LB agent applies at time t a split ratio x_p^k for each tunnel k over each path $p \in \mathbb{P}_k$ ($x_p^k \in [0, 1]$ and $\sum_{p \in \mathbb{P}_k} x_p^k = 1 \ \forall k \in K$).

The delay on each path p for a tunnel k is denoted d_p^k and the tunnel delay, denoted d^k is computed as follows:

$$d^k = \max_{p \in \mathbb{P}_k} \quad d_p^k \tag{1}$$

In practice, d_p^k is continuously measured by access routers.

Problem formulation. The main objective is to derive an optimal load balancing policy so that the SD-WAN overlay delivers the best QoS. In the rest of the paper, we consider as primary target the minimization of the average tunnel delay under the constraint that link capacity constraints are not violated (safety constraint). Indeed, this safety measure prevents that 1) congestion is induced by miss configured split ratios and that 2) the average tunnel delay is not artificially minimized by rejecting traffic (i.e., by intentionally creating some congestion). To prevent high link delays, or very heterogeneous ones even if the average delay is low, a common practice is to enforce a Maximum Link Utilization (MLU) $\mu \in [0, 1]$ over all links. LB agents at headquarter and branches are configured by a network controller.

In this case, load balancing policies, which are derived according to dynamic tunnel traffic T^k, solve the following optimization problem:

$$\min_{x_p^k} \quad \frac{\sum_{k \in K} d^k}{K} \tag{$\mathcal{P}$}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{i=0, p \in \mathbb{P}_k}^{|p|} T^k . x_i^k \leq \mu . c_e \qquad \forall e \in E \tag{$\mathcal{C}_0$}$$

$$\sum_{i=0}^{|p|} x_i^k = 1 \qquad \forall x_i^k \in [0, 1] \tag{$\mathcal{C}_1$}$$

where problem ($\mathcal{P}$) minimizes the average tunnel delay. Constraints ($\mathcal{C}_0$) guarantee that the traffic over each edge e in the network is kept under the MLU. Constraints ($\mathcal{C}_1$) ensure that splits ratios sum to 1.

IV. Learning-based and safe load balancing

In this section, we propose a constrained policy optimization for load balancing based on Deep Reinforcement Learning (DRL) algorithms and a Control Barrier Function (CBF) [13].

A. Learning-based optimization

Our optimization problem can be formulated as a Markov Decision Process (MDP) which is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$ where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ is the set of available actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function which gives the reward for the transition from one state to another given an action, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition matrix, which gives the probabilities of transitioning from one state to another given an action and $\gamma \in [0, 1]$ is the discount factor. A policy $\pi : S \rightarrow A$ refers to the probability of taking an action $a \in \mathcal{A}$ under state $s \in \mathcal{S}$. The agent iteratively interacts with the (networking) environment to learn an optimal policy π^* that chooses actions with the best payoff. To solve the MDP associated with problem ($\mathcal{P}$), a centralized controller is employed in the network. The controller can periodically collect at every time t information from the different AR routers: the traffic demand T^k, the delay d^k of each tunnel $k \in K$ and the maximum link utilization μ. In this context, we consider the observation space or state s_t as the set of traffic demands T^k for all tunnels $k \in K$. The action space is determined as the set of split ratios x_p^k for all paths $p \in \mathbb{P}_k$ of each tunnel $k \in K$.

By using RL algorithms, we learn the optimal stochastic control policy $\pi(a|s)$ that maximizes the performance measure (i.e., $J(\pi)$) which is expressed as follows:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_t^{\infty} \gamma^t r_t(s_t, a_t) \right] \tag{2}$$

where $\tau \sim \pi$ denotes a trajectory during which actions are sampled according to the policy $\pi(a|s)$.

Given the fact that policy gradient RL methods have shown good performance in continuous control problem [22] [23], in the rest of this paper, we consider off-policy learning (e.g. Deep Deterministic Policy Gradient (DDPG), Twin-Delayed Deep Deterministic Policy Gradient (TD3)) and on-policy learning (e.g., Proximal Policy Optimization (PPO)) algorithms. In the former technique, actions are sampled to encourage the agent to explore the environment. Then, a target deterministic policy is derived from these actions, supported by the actor-critic architecture [22] with a replay buffer [24]. On the other hand, on-policy methods iteratively derive the target policy based on samples obtained from their own previous actions and the update is carried out by solving the following optimization problem:

$$\pi_{i+1} = \underset{\pi}{argmax} \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A^{\pi_i}(s, a) \tag{3}$$

$$\text{s.t.} \quad KL \left[\pi_{i+1}(.|s_t), \pi_i(.|s_t) \right] \leq \delta_{KL} \tag{4}$$

where $A^{\pi_i}(s, a)$ is the advantage of performing action a (sampled by old policy π_i) at state s, $\rho_{\pi_i}(s)$ represents the frequency of visit of state s under policy π_i and $KL \left[\pi_{i+1}(.|s_t), \pi_i(.|s_t) \right]$ refers to Kullback-Leibler divergence between new policy and old policy [25] and this value is bounded by a target δ_{KL}. Both on and off-policy gradient algorithms do not consider safety, therefore our goal is to complement these model-free learning algorithms with a Control Barrier Function (CBF) which guarantees safety during exploration and exploitation.

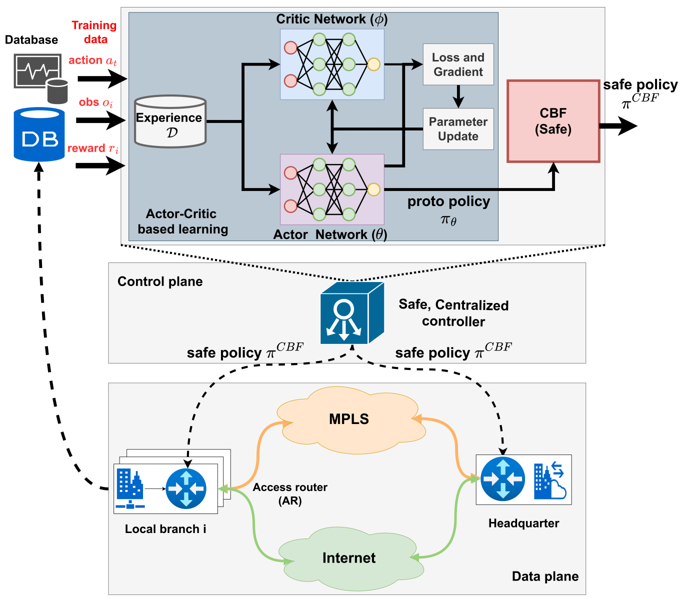


Fig. 2: Safety-based actor-critic learning architecture.

Figure 2 introduces the control plane architecture where each LB agent is centrally configured by the controller. The controller can periodically query the network state (i.e., tunnel delays, OD traffic, MLU) at each LB and its control system is composed by two main blocks that focus on policy optimization and safety, respectively. The former block, which is based on an actor-critic architecture [22], learns the (unconstrained) optimal policy. Accordingly, the value function and the policy are approximated using two different neural networks, parameterized by ϕ and θ, respectively. Each learning parameter is responsible for either maximizing the actor's objective function (i.e., $\mathcal{J}_b^A$) or the critic's loss function (i.e., $\mathcal{L}_b^C$). The specific expressions for $\mathcal{J}_b^A$ and $\mathcal{L}_b^C$ heavily depend on the off/on policy reinforcement learning algorithms being used. For instance, the explicit functions for DDPG (off-policy) and PPO (on-policy) algorithms can be respectively found in [22] and [26]. The details of the optimization procedure are shown in Algorithm 1.

In order to enforce safe exploration during learning and safe policy execution during testing, a safety block, based on a CBF function [21], is implemented on top of DRL algorithms.

B. *Safe policy exploration and exploitation*

The CBF function serves as a projector to convert the *proto-policy* π_θ, which is the parameterized actor network from which unsafe actions might cause congestion, into a *safe policy* π^{CBF}. Its main objective is to guarantee that the MLU μ remains below 1 (i.e., $\mu(a^{CBF}) \leq 1, \forall a^{CBF} = \pi^{CBF}(.|s)$). As illustrated by Figure 3, this projection maintains the safe policy π^{CBF} as close as possible to the proto policy and keeps load balancing system in safety condition. Safe policy projection

Algorithm 1 RL-based optimization algorithm

```
Initialize parameterized RL policy $\pi_{\theta_0}$
Initialize value function parameter $\phi_0$
Define episode length $L_{eps}$, batch size $B$, total episodes $N_{eps}$
Initialize experience array $\mathcal{D} = \{\emptyset\}$
if episode $k < N_{eps}$ then
    for time step $t = 1, ..., L_{eps}$ do
        Observe state $s_t$
        Sample action $a_t \sim \pi_{\theta_k}(.|s_t)$
        Perform local search algorithm
            $a_t^{CBF} = Local\_Search(s_t, a_t)$
        Deploy action $a_t^{CBF}$ and obtain reward $r_t$, next state $s_{t+1}$
        Store experience tuple $< s_t, a_t^{CBF}, s_{t+1}, r_t >$ in $\mathcal{D}$
    end
    for batch sampling $B$ experiences in $\mathcal{D}$ do
        Perform gradient ascent on actor network
            $\theta_{k+1} \leftarrow \frac{1}{|B|} \arg\max_{\theta_k} \sum_{b=0}^{|B|} \mathcal{J}_b^A(\theta_k)$
        Perform gradient descent on critic network
            $\phi_{k+1} \leftarrow \frac{1}{|B|} \arg\min_{\phi_k} \sum_{b=0}^{|B|} \mathcal{L}_b^C(\phi_k)$
    end
end
return parameterized policy $\pi_\theta$, parameterized value function $\phi$
```

is performed at the centralized controller in both learning and testing. Following the safe policy π^{CBF}, each CBF action is determined according to the following optimization problem:

$$\begin{aligned} & a^{CBF} = \underset{a_t^{CBF}}{argmin} \left\| a_t^{CBF} - a_\theta \right\|_1 \\ s.t. \quad & a^{CBF} \in \mathcal{A} \\ & \mu(a^{CBF}) \leq 1 \end{aligned} \tag{5}$$

With respect to reward function design, we highlight that the objective function of problem ($\mathcal{P}$) is enough to minimize the average tunnel delay in the network. However, as mentioned in Section III, when only minimizing the average delay, the system may induce large delays for a small number of tunnels due to a high link utilization or congestion, i.e. packet drops to

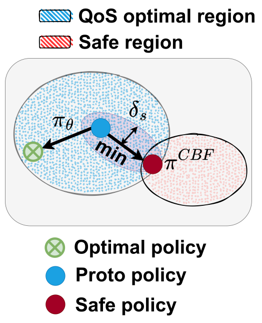


Fig. 3: From proto-policy to safe policy with CBF.

further reduce the average delay. Therefore, in order to make the DRL agent also aware of link utilization, we design our reward function to be the weighted sum of the average delay and the MLU (i.e., μ) as given by Equation 6.

$$r_t(s_t, a_t) = -\sigma \frac{\sum_{k\in K} d_{k,t}}{|K|} - (1-\sigma)\mu \qquad (6)$$

where $\sigma \in [0, 1]$ emphasizes the importance of the average tunnel delay over a low MLU. While this reward cannot guarantee itself a hard safety, it guides RL agent in learning a policy which is both QoS optimal and safe after convergence. To ensure anytime and hard safety, we explain in the following how we incorporate a CBF function on top of DRL algorithms to constrain the exploration and exploitation so that the MLU does not exceed 1 (i.e., $\mu \leq 1$).

CBF function. The CBF function, applied on top of DRL algorithms, brings two main benefits. Firstly, it guarantees that any action, which are stochastically sampled from the parameterized actor network, will not cause a during both exploration and exploitation. Secondly, it considerably contributes to obtaining a better reward-per-learning-step as the MLU is part of the reward. Indeed, by correcting "bad" proto-actions, which produce a high MLU and overloaded links, the output safe action results in a better or at least equal MLU compared to the initial proto-action. It leads to a fast convergence towards an optimal safe policy. For the projection of proto-actions into safe actions, we propose to use as a CBF function the local search algorithm detailed in Algorithm 2.

In principle, given a proto-action, the CBF stochastically attempts to generates N safe actions within a neighborhood of radius δ_s. The generation of actions is based on three different policies where the information with regards to the link utilization of each path p in the tunnel k is exploited:

- ***Naive policy***: This policy randomly picks any tunnel $k \in K$. For each selected tunnel, a random value $\epsilon \sim Uniform(0, \delta_s)$ is added/subtracted to current split ratios given by the proto-action on each path $p \in \mathbb{P}_k$. In particular, the traffic load on the highest utilization path p of the selected tunnel k will be reduced by an amount of $\epsilon.T^k_{t,p}$. This traffic amount will be equally distributed to the remaining paths of the tunnel. In our scenario where there are only 2 paths per tunnel, the reduced traffic on the path with higher utilization will be directly transferred to the path with lower utilization.
- ***DeltaUtil policy***: This policy selectively focuses on tunnels where the difference between their highest path utilization and lowest path utilization is greater than a certain threshold. In our case, a threshold on the difference of 50 % is chosen. Once this criteria is met, a randomly generated value $\epsilon \sim Uniform(0, \delta_s)$ is used to fine-tune split ratios in the proto-action, similarly to the **Naive** policy.
- ***MaxUtil policy***: This new policy inherits the principles of the ***DeltaUtil policy*** policy, but it uses a different criteria for selecting tunnels. Specifically, any tunnel k that has a path load utilization above a threshold of 100 % (e.g., $\exists p \in \mathbb{P}_k \mid \mu_p \geq 1$, which is unsafe) will be the target for proto-action modification. Once the set of congested tunnels is determined, each random value $\epsilon_k \sim Uniform(0, \delta_s)$ will be used to adjust the initial split-ratio of each selected tunnel k, which is primarily decided by the proto-policy. As a consequence, an amount of traffic $\epsilon_k.T^k_{t,p}$ will be withdrawn from the path p of tunnel k with the highest path utilization, and added to remaining paths.

Algorithm 2 CBF based on Local Search algorithm

```
Returned action from RL agent $a^{RL}_\theta$
Local search radius $\delta_s$
CBF solutions $N$
Local search policy $\pi^{CBF}$
Local search max iteration $M$
Feasible solution $feas\_sol = \{\}$
if $\mu(a^{RL}_\theta) \leq 1$ then
    $a^{CBF} = a^{RL}_\theta$
else
    for $m$ in $M$ do
        for $n$ in $N$ do
            Randomly sample $\epsilon_n \sim Uniform(0, \delta_s)$
            Stochastic action generation $a^{CBF}_n \sim \pi^{CBF}$
            Perform action update: $a^{CBF}_n = a^{RL}_\theta \pm \epsilon_n$
            if $\mu(a^{CBF}_n) \leq 1$ then
                Append $a^{CBF}_n$ to $feas\_sol$
            end
        end
    end
    if $feas\_sol \neq \oslash$ then
        $a^{CBF} = \underset{a^{CBF}_i \in feas\_sol}{argmin} \left\| a^{CBF}_i - a^{RL}_\theta \right\|_1$
    else
        $a^{CBF} = \underset{n}{argmin} \quad \mu\{a^{cbf}_n\}$
    end
end
End
```

After generating a large number of actions around a proto action, a feasible action (i.e., MLU is below 100%) is selected in such a way that its distance to the original proto action is the smallest. The returned action is quasi-guaranteed to be safe and helps learning safe policies. As local search policies are heuristic, it may be that they cannot correct a proto-action for which a safe action exists. In this case, the CBF action, which returns the lowest MLU, will be selected.

Our local search algorithm is important in both training and exploitation phases. In the training phase, it helps to ensure that the network learns safe policies by eliminating the possibility of taking actions that could cause safety issues (i.e., violation of link capacity constraint). In the exploitation phase, it helps to ensure that the network continues to operate safely even under unseen network conditions.

V. Results and discussions

We now evaluate the performance of the DRL-CBF solution on the SD-WAN scenario presented in Section III.

A. Network environment

Our simulations are carried out using Python's Gym toolkit [27], which is a well-known Application Programming Interface (API) for interfacing between our (safe) learning algorithms and our Python-based SD-WAN environment.

In order to demonstrate the stochastic traffic behavior at each *OD flows*, we generate noisy sinusoidal traffic to model diurnal variations, as shown by Figure 4. The phase of each OD flow is shifted to make incoming traffic to each site somewhat variable and likely to cause congestion at bottleneck links without appropriate load balancing.

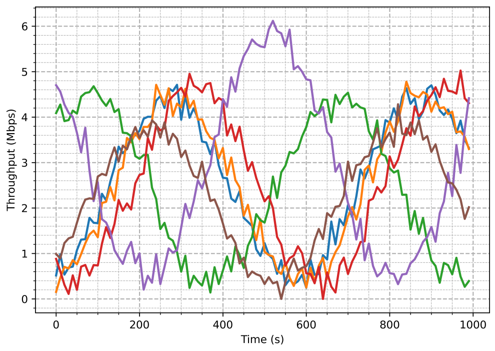


Fig. 4: Example of tunnels' traffic over a window of 1000s.

We adopt a simple M/M/1 queuing model to compute the delay on each *link* e and considers a propagation delay d_{prop}. The link delay d_e is then derived as follows:

$$d_e = d_{prop} + \frac{1}{c_e - l_e} \tag{7}$$

In our scenario, each MPLS and Internet link has a fixed capacity of $c_e = 6Mbps$ and $c_e = 15Mbps$, respectively and d_{prop} represents the propagation delay.

The delay on each path p of a tunnel k (i.e. d_p^k) is then calculated as the sum of the delay on the edge that is involved in that path as follows:

$$d_p^k = \sum_{e \in p, p \in \mathbb{P}_k} d_e \tag{8}$$

Furthermore, in order to emulate the behavior of TCP in the considered network, we adopted a min-max fairness rate allocation policy using a standard water-filling algorithm [28].

B. Algorithms implementation

Relying on Stable-Baseline3 [29], a well-known library of RL algorithms, we have exploited two different types of RL algorithms: off-policy with DDPG [30]) and on-policy with PPO [26]). Given that they do not guarantee safety, we have applied the CBF function based on local search algorithms (Algorithm 2) on top of them.

We implemented the solution on a server composed by a CPU Intel® Xeon® Platinum 8164 (104 logical cores and 1024 GB of RAM memory) and a GPU NVIDIA® Tesla V100 (5120 CUDA cores and 32 GB of DRAM), as shown in Figure 5. As local search algorithms can be massively parallelized, we

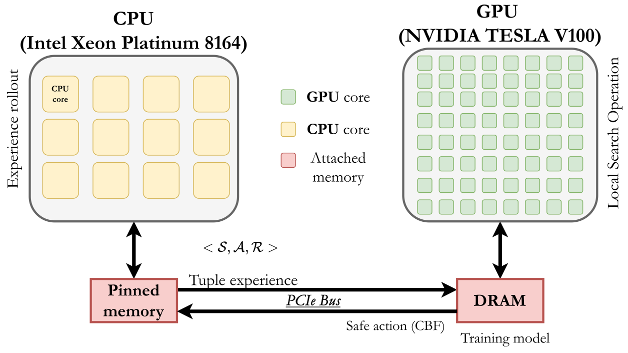


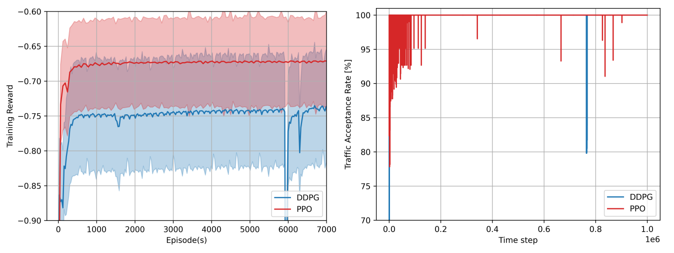
Fig. 5: System architecture with 1) network environment running on CPU and 2) safe RL algorithms on GPU.

TABLE I: Simulation (hyper)parameters

Simulation (Hyper) parameters	Value	
	Off-policy Algo (DDPG)	On-Policy Algo (PPO)
Gradient clipping maximum	■	0.5
Clipping parameter ε	■	0.2
Target KL divergence δ_{KL}	■	0.03
Delayed network update rate	0.05	■
Learning rate	1e-5	
Discounted factor γ	0.7	
Hidden layers	3	
Dimension of hidden layer	512	
Batch size $\|B\|$	256	
Frequency of model updates (steps)	256	
Episode length L_{eps} (steps)	128	
Max local search iteration (M)	20	
Local search solutions (N)	1.000	
Local search radius (δ_s)	0.3	
Reward parameter σ	0.8	
Training steps	1.000.000	

implemented them with CUDA libraries so that they fully benefit from all GPU cores available. Therefore, model training and intensive local search algorithms are fully performed at GPU side. Once a safe policy is found, it is transferred to the CPU, where our SD-WAN environment is located, to perform a rollout step. The resulted tuple experiences are then moved back to GPU for further model training.

Table I enumerates the list of parameters that we applied. To stabilize the training process in DDPG, delayed network updates have been used so that the target actor/critic networks are updated less frequently than the main actor/critic by a factor of 0.05. The clipped version of PPO [26] has been used with a clipping parameter $\varepsilon = 0.2$ and a target KL divergence set at $\delta_{KL} = 0.03$. To avoid destructively large weight updates, the gradient is also clipped at 0.5. Besides, a fully connected neural network with 3 layers of 512 neurons is used to approximate policy and value functions. With regards to local search algorithms, we take the best solution out of $N * M = 20.000$ points generated around each unsafe action returned by the DRL agent in a radius of $\delta_s = 0.3$. Finally, our models are updated after each rollout of 256 steps, which is equivalent to 2 episodes, and the learning process is studied in $t_s = 1.000.000$ training steps.



(a) Episodic training rewards (b) Traffic acceptance rate

Fig. 6: Episodic training reward and traffic acceptance rate at learning phase.

To obtain an optimal solution as benchmark, the following constraints ($\mathcal{C}_2$) are added to Problem $\mathcal{P}$ so that the link delay is computed according to M/M/1 queuing model. The resulting Non Linear Problem (NLP) is solved by the SCIP [14] solver. Note that this constraint plays an important role for our benchmark but, in practice, it is not explicitly known because performance might not follow the M/M/1 model.

$$d_e \geq \frac{1}{c_e - \sum_{k \in K} \sum_{p \in \mathbb{P}_k} l_p^k} \quad \forall e \in E \qquad (\mathcal{C}_2)$$

C. Obtained results

On baseline without safety. Figure 6a compares the average training reward during each episode for DDPG and PPO, respectively. This figure highlights that PPO typically achieves better rewards than DDPG. Besides, the DDPG agent tends to be trapped into local optimum policies, which can lead to sub-optimal performance. In terms of safety, Figure 6b illustrates the percentage of the total traffic that is accepted due link capacity constraints. Both algorithms encourage policy exploration at early stages in training, which accidentally causes link congestion and traffic rejection. A severe unsafe attempt of policy exploration for DDPG can be observed at episode 6000. However, even if it helps learning better policies, unsafe exploratory actions should be avoided in production systems, especially for on-policy algorithms where the model is updated and applied after a smaller number of episodes.

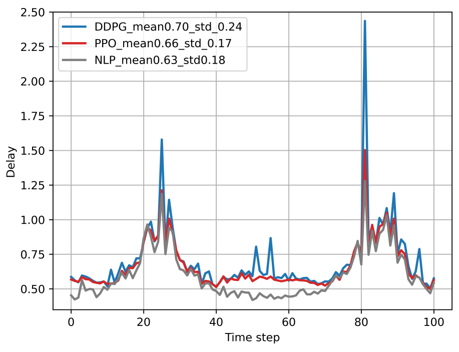
In order to compare performance during testing, our learning models using DDPG and PPO algorithms are selected after the last episode training. At this point, training models are well aware of delay and MLU minimization in the objective function. Therefore, traffic rejection due to link capacity violation is unlikely to happen. To define a testing scenario, we generated 100 traffic samples for each OD tunnel following the traffic pattern presented in Figure 4. As illustrated in Figures 7a and 7d, delay and MLU of learning-without-safety models are compared to the optimal NLP solution obtained with SCIP. They reveal that near-optimal delays are obtained using conventional DDPG and PPO learning algorithms. Besides, the MLU during testing is safely kept below 1, resulting in no traffic rejection. Furthermore, network delay is better for PPO compared to DDPG since the local optimum trap is not observed during training.

With safety. When the safety CBF layers are applied on top of current off/on policy learning algorithms, testing performance results are depicted in Figures 7b, 7e, 7c and 7f, respectively. With respect to safety, off-policy learning using DDPG, Figures 7b and 7e demonstrate that the DDPG-CBF agent does not handle delay well, especially when the total traffic demand is high. In particular, many high delay peaks are observed during testing phase regardless CBF policies, although safety is always respected (MLU belows 1). PPO-CBF significantly improves network delay and better controls the MLU in testing phase as illustrated by Figures 7c and 7f. Furthermore, no capacity violations happen during learning as indicated by Figure 8c. These results suggest that, with safety, on-policy learning outperforms off-policy learning as near-optimal performance is obtained and hard safety requirements are met.

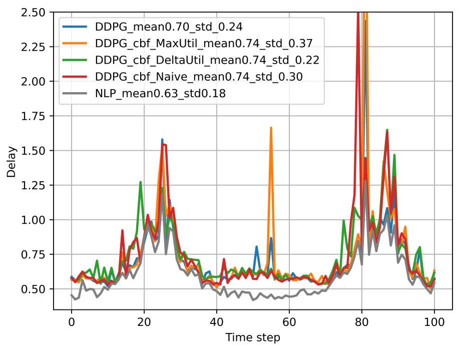
In addition, Figure 8c also shows the training performance for the three CBF functions we implemented. As we can see, traffic acceptance during learning is significantly improved compared to the case without CBF, as illustrated in Figure 6b. Moreover, the **MaxUtil** policy outperforms **Naive** and **DeltaUtil** policies in achieving no traffic rejection during learning. Indeed, this policy directly targets tunnels that make the network congested. With respect to testing performance, Figure 7b, 7e, 7c and 7f show that all three CBF functions nearly get the same performance and that PPO-CBF performs much better than DDPG-CBF. Indeed, no delay spikes and smoother performance are observed for PPO-CBF.

To shed the light on the convergence of training rewards with and without CBF functions, Figure 8a and 8b compare the episodic training rewards of DDPG-CBF and PPO-CBF, respectively. These figures reveal that on-policy learning is faster and stable compared to off-policy learning. Although DDPG-CBF achieves a better reward than its non-safe version, its training curve is slightly unstable compared to the training curves of PPO-CBF. As a result, combining safe learning with this off-policy approach is more challenging.

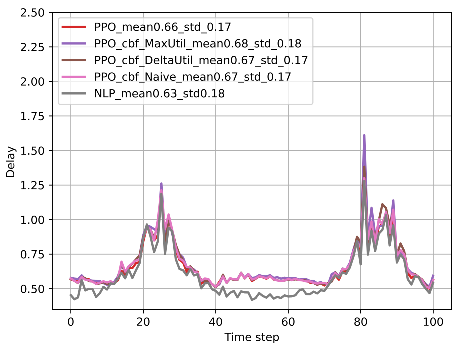
Figure 8d demonstrates the benefits of our hardware architecture illustrated on Figure 5 for training acceleration. We compare the average, real execution time required to successfully perform one iteration of Algorithm 1, which includes local search algorithm and model update, with and without (i.e. CPU) using the GPU. It can be observed that the GPU implementation significantly speeds up calculation and training time. In particular, a feasible (safe) action is found in less than $0.1(s)$ when using GPU, compared to more than $11(s)$ using CPU. This significant time saving favors the usage of our methods in practice, since the model can also be updated every 256 steps in roughly 25.6 seconds, rather than 2816 seconds using only CPU. As a result, our model can be fully trained in one day using 1.000.000 training steps instead of 134 days when using the CPU. This time acceleration has a significant impact on the possibility to deploy our solution in practice. Indeed, since network measurements (e.g., delay,



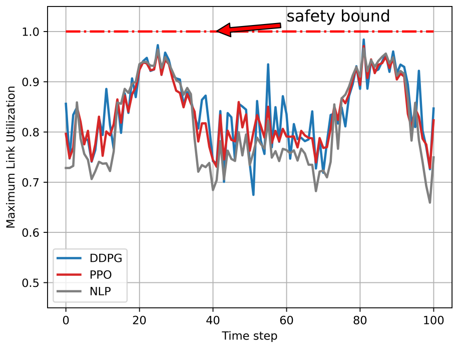
(a) Delay results of DDPG, PPO and NLP



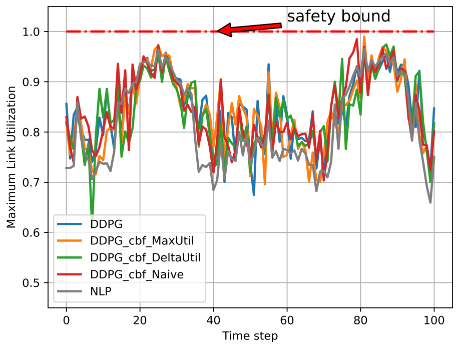
(b) Delay results of DDPG-CBF and NLP



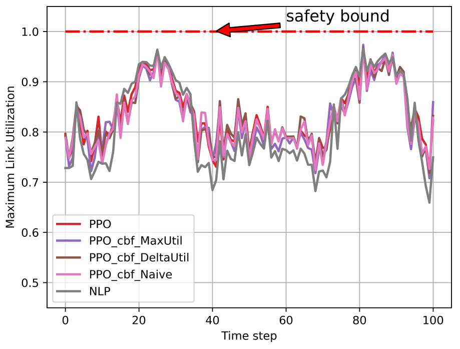
(c) Delay results of PPO-CBF and NLP



(d) MLU results of DDPG, PPO and NLP

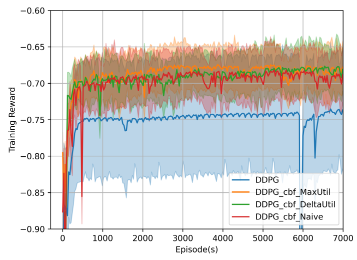


(e) MLU results of DDPG-CBF and NLP

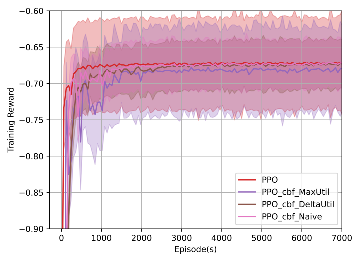


(f) MLU results of PPO-CBF and NLP

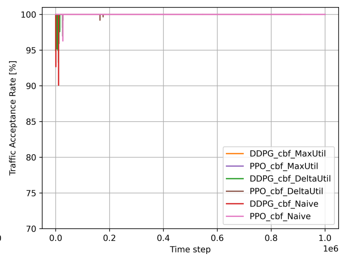
Fig. 7: Testing performance on average tunnel delay and MLU.



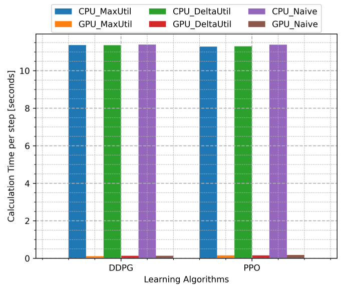
(a) Reward of DDPG-CBF



(b) Reward of PPO-CBF



(c) Traffic acceptance rate



(d) Calculation time

Fig. 8: Training performance: (a,b) average episodic training reward, (c) traffic acceptance rate and (d) average computation time per step.

traffic, loss, etc.) can be collected every 1s, on-policy leaning models can be updated every 256s in 25s, which is reasonable.

VI. Conclusion

We presented a novel approach combining the Deep Reinforcement Learning (DRL) and a Control Barrier Function (CBF) to guarantee safe exploration and exploitation in the context of Software Defined-Wide Area Network (SD-WAN). Tackling a typical load balancing problem where latency needs to be optimized while meeting safety requirements in terms of capacity constraints, our DRL-CBF approach is able to achieve near-optimal performance with safe exploration and exploitation. Furthermore, we show that on-policy optimization based on PPO achieves better performance than off-policy learning with DDPG. We implemented all the algorithms on GPU to accelerate training by approximately 110x times and achieve model updates for on-policy methods within a few seconds, making the full solution practical.

Future works along these lines include the integration with a network simulator and a testbed for a more realistic performance evaluation. We also plan to address more challenging environments where, for instance, QoE needs to be optimized and other constraints need to be handled.

Acknowledgment

This work was partially supported by the French Nation Research Agency (ANR) SAFE project under grant ANR-21-CE25-0005.

References

[1] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, “Software-Defined Wide Area Network (SD-WAN): Architecture, Advances and Opportunities,” in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[2] Y. Magnouche, P. T. A. Quang, J. Leguay, X. Gong, and F. Zeng, “Distributed Utility Maximization From the Edge in IP Networks,” in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 224–232.

[3] P. T. A. Quang, S. Martin, J. Leguay, X. Gong, and X. Huiying, “Intent-Based Routing Policy Optimization in SD-WAN,” in *ICC 2022-IEEE International Conference on Communications*, 2022, pp. 4914–4919.

[4] W. Ben-Ameur and A. Ouorou, “Mathematical Models Of The Delay Constrained Routing Problem,” *Algorithmic Operations Research*, vol. 1, no. 2, 2006.

[5] L. Kleinrock, *Communication Nets: Stochastic Message Flow And Delay*. Courier Corporation, 2007.

[6] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, “Experience-driven networking: A Deep Reinforcement Learning Based Approach,” in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, p. 1871–1879.

[7] T. Mai, H. Yao, Z. Xiong, S. Guo, and D. T. Niyato, “Multi-Agent Actor-Critic Reinforcement Learning Based In-network Load Balance,” in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[8] M. Kim, M. Jaseemuddin, and A. Anpalagan, “Deep Reinforcement Learning Based Active Queue Management For Iot Networks,” *Journal of Network and Systems Management*, vol. 29, no. 3, p. 34, 2021.

[9] O. Houidi, D. Zeghlache, V. Perrier, P. T. A. Quang, N. Huin, J. Leguay, and P. Medagliani, “Constrained Deep Reinforcement Learning For Smart Load Balancing,” in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 207–215.

[10] H. Fawaz, O. Houidi, D. Zeghlache, J. Lesca, P. T. A. Quang, J. Leguay, and P. Medagliani, “Graph Convolutional Reinforcement Learning For Load Balancing And Smart Queuing,” in *2023 IFIP Networking Conference (IFIP Networking)*. IEEE, 2023, pp. 1–9.

[11] S. Troia, F. Sapienza, L. Varé, and G. Maier, “On Deep Reinforcement Learning for Traffic Engineering in SD-WAN,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2198–2212, Jul. 2021.

[12] Q. Lin, Z. Gong, Q. Wang, and J. Li, “RILNET: A Reinforcement Learning Based Load Balancing Approach for Datacenter Networks,” in *ML for Networking*, 2019.

[13] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, “End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks,” no. arXiv:1903.08792, Mar. 2019.

[14] K. Bestuzheva, M. Besançon, and et al., “The SCIP Optimization Suite 8.0,” no. arXiv:2112.08872, Dec. 2021.

[15] N. Bouacida and B. Shihada, “Practical and Dynamic Buffer Sizing Using LearnQueue,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1885–1897, Aug. 2019.

[16] A. Y. Kamri, P. T. A. Quang, N. Huin, and J. Leguay, “Constrained Policy Optimization for Load Balancing,” in *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*, Apr. 2021, pp. 1–6.

[17] C. Tessler, D. J. Mankowitz, and S. Mannor, “Reward Constrained Policy Optimization,” *Computing Research Repository (CoRR)*, vol. abs/1805.11074, 2018.

[18] X. Zhang, Y. Xu, Y. Chen, and D. Li, “Path Planning Model Based on Constrained Policy Iteration,” in *2022 41st Chinese Control Conference (CCC)*, 2022, pp. 5518–5523.

[19] M. Huang and J. Chen, “Proactive Load Balancing Through Constrained Policy Optimization for Ultra-Dense Networks,” *IEEE Communications Letters*, vol. 26, no. 10, pp. 2415–2419, Oct. 2022.

[20] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained Policy Optimization,” *Computing Research Repository (CoRR)*, vol. abs/1705.10528, 2017.

[21] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control Barrier Functions: Theory And Applications,” *Computing Research Repository (CoRR)*, vol. abs/1903.11199, 2019.

[22] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML’14. Beijing, China: JMLR.org, Jun. 2014, pp. I–387–I–395.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., ser. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.

[24] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight Experience Replay,” *Computing Research Repository (CoRR)*, vol. abs/1707.01495, 2017.

[25] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms.” *Computing Research Repository (CoRR)*, vol. abs/1707.06347, 2017.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *ArXiv*, vol. abs/1606.01540, 2016.

[28] D. P. Bertsekas and R. G. Gallager, *Data Networks*. Prentice Hall, 1992.

[29] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[30] 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,” no. arXiv:1509.02971, Jul. 2019.