Policy Distillation for Real-time Inference in Fronthaul Congestion Control

JEAN P. MARTINS¹, IGOR ALMEIDA¹, RICARDO SOUZA¹, SILVIA LINS¹
¹Ericsson Research. (e-mail: {jean.martins, igor.almeida, ricardo.s.souza, silvia.lins}@ericsson.com)

Corresponding author: Silvia Lins (e-mail: silvia.lins@ericsson.com).

ABSTRACT
Centralized Radio Access Networks (C-RANs) are improving their cost-efficiency through packetized fronthaul networks. Such a vision requires network congestion control algorithms to deal with sub-millisecond delay budgets while optimizing link utilization and fairness. Classic congestion control algorithms have struggled to optimize these goals simultaneously in such scenarios. Therefore, many Reinforcement Learning (RL) approaches have recently been proposed to deal with such limitations. However, when considering RL policies’ deployment in the real world, many challenges exist. This paper deals with the real-time inference challenge, where a deployed policy has to output actions in microseconds. The experiments here evaluate the tradeoff of inference time and performance regarding a TD3 (Twin-delayed Deep Deterministic Policy Gradient) policy baseline and simpler Decision Tree (DT) policies extracted from TD3 via a process of policy distillation. The results indicate that DTs with a suitable depth can maintain performances similar to those of the TD3 baseline. Additionally, we show that by converting the distilled DTs to rules in C++, we can make inference-time nearly negligible, i.e., sub-microsecond time scale. The proposed method enables the use of state-of-the-art RL techniques to congestion control scenarios with tight inference-time and computational constraints.

INDEX TERMS Reinforcement learning, policy distillation, congestion control, real-time inference, fronthaul networks.

I. INTRODUCTION
With the evolution of 5th Generation Mobile Networks (5G), research on more adaptable network architectures is gaining momentum targeting cost reduction and increased user satisfaction. Centralized Radio Access Networks (C-RAN) play a fundamental role in this context, being envisioned for increased flexibility and lower cost of deployment [1]. The C-RAN architecture splits the regular radio base station into two different nodes: (1) the Baseband Unit (BBU) and (2) the Remote Radio Unit (RRU). Although the split reduces the complexity of RRU nodes, it also introduces (very) high throughput demands to fronthaul links, reaching nearly 50 times higher demands than backhaul [2] (see Figure 1).

One alternative to meet such requirements while improving the cost-efficiency of transport networks is migrating from dedicated fiber links (with Common Public Radio Interface (CPRI) [3] protocol) towards more flexible, packetized network deployments, which can benefit from statistical multiplexing. However, this shared infrastructure scenario also brings additional challenges to the transport network deployments since fronthaul links could now suffer from network congestion due to, for example, aggressive radio schedulers.

Both congestion control (CC) and Flow Control (FC) algorithms work at the flow level, where a flow consists of a sequence of packets from a particular source-destination pair. While FC prevents a faster host from overwhelming a slower receiver, CC manages the transmission rate of multiple senders to avoid network overload. There are several well-
established CC algorithms in the literature, and primarily for backward-compatibility reasons, solutions based on Transmission Control Protocol (TCP) are still widely adopted. However, classic CC algorithms have also shown some limitations when dealing with low-latency and high throughput scenarios, such as those required by fronthaul networks [4].

Many authors have applied learning-based techniques to improve TCP congestion control, with promising results being reported [5]-[10]. The range of proposals goes from optimization heuristics, such as Remy [11] that searches for the best congestion window size and intersend time based on the current state of the network; to classical Reinforcement Learning (RL) based on tabular Q-learning using Sparse Distributed Memories to approximate state-action values [12]. More recent proposals mainly rely on deep RL algorithms, be it learning congestion control policies from the scratch [13]-[20], or hybrids that learn from or cooperate with classical algorithms [21], [22]. Most of these works report improvements in comparison to classical TCP algorithms [11], [12], [18], mainly in metrics such as fairness among senders, latency and overall link utilization.

The fast-growing number of RL-based congestion control proposals indicates a trend similar to the RL area as a whole. The maturity of the results already requires the investigation of more practical aspects related to the deployment of RL agents to the real world. Most of the challenges to real-world RL are also valid to RL-based congestion control [23], [24], but some are more evident as discussed in [21], [25]. Among challenges such as sample efficiency, delayed rewards, delayed observations, and delayed actuation, etc., this paper focuses on dealing with real-time inference aspects crucial in fronthaul networks.

The congestion control loop in fronthaul networks is high-speed (microsecond) due to its low latency and high throughput characteristics, a challenge regarding inference-time. Most RL-based proposals in the literature focus on internet congestion control, where the Round Trip Times (RTTs) are much larger, allowing for higher inference times. However, even in such scenarios, the existing methods usually rely on time windows in which the agents propose their actions. Such strategies are known to deteriorate the reactivity of the policies [21], an issue that gets worse in low-latency scenarios and is often tackled with ACK-based CC policies, which can actuate on the transmission rate at every ACK received.

This paper evaluates the use of policy distillation as a way to extract from a complex RL policy, simpler models whose inference-time is reduced, without considerable performance deterioration. The method consists of training a Twin-Delayed Deep Deterministic Policy Gradient (TD3) policy to act as a baseline and then via policy distillation producing Decision Trees (DTs) of different sizes. A comprehensive set of experiments evaluates the performance of the original TD3 policy, its distilled counterparts, and their inference-time, using both Python and C++ versions of the policies.

The paper is organized as follows. Section II provides a background on RL and congestion control. Section III describes the training environment with its underlying learning problem, technical aspects of the network simulations, and the policy distillation process. Finally, Section IV describes the experiments and results for different network configurations, and Section V presents our concluding remarks.

II. BACKGROUND

This section defines fundamental aspects of Reinforcement Learning (RL), along with some representative algorithms in the area. A brief literature review of RL applied to congestion control scenarios is also put into perspective.

A. REINFORCEMENT LEARNING

RL comprises a set of techniques that enable an agent to learn to interact with an environment by iteratively exploring and evaluating the outcomes of its actions. The overall goal in RL is to derive policies that map the observed situations experienced by the agent to actions that would maximize the cumulative rewards received by the agent [26].

The formalism provided by Markov Decision Processes (MDPs) is a useful tool for defining the RL framework. From such a perspective, the specification of an environment for a particular task of interest consists of specifying four elements $(S, A, r, Pr)$. The state-space $S$ determines what information about the system is available to the agent, the action space $A$, defines how the agent can interact with the system, and the reward function $r : S \rightarrow \mathbb{R}$, measures the immediate quality of such interactions. Optionally, a state-transition probability distribution $Pr$ defining the dynamics of the environment can also be specified.

At every timestep $t = 1, 2, \ldots$, the agent observes the current state of the system $s_t \in S$ and by consulting its policy decides for an action $a \in A$ that maximizes the expected future rewards. RL concerns in learning such policies from the data obtained from the interactions between agent and environment.

In general, the optimality of a policy $\pi$ depends on the states an agent visits as it interacts with the environment. A sequence of states is usually called a trajectory $\tau$, and from it, we can define an optimal policy as

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau)].$$  \hspace{1cm} (1)

where $R(\tau) = \sum_{s \in \tau} r(s)$ refers to the cumulative rewards received during the trajectory $\tau$.

There are different RL techniques to deal with the many aspects that arise in practice. In some environments, the agents must choose from a finite set of actions, while in others, they face nearly infinite-sized action spaces. The same is true for the state spaces, ranging from small grid-worlds to continuous state spaces. In both cases, models for function approximation have become a requirement to deal with real-world environments, a role mainly fulfilled by Deep Neural Networks (NNs), hence Deep Reinforcement Learning (DRL).

$^1$Undiscounted infinite-horizon, for alternative formulations see [26].
1) Discrete action spaces

**Q-learning** is a classic RL algorithm for small-scale discrete state- and action-spaces [26]. Such a technique works by bootstrapping estimates of a function \( Q(s, a) \) as the agent interacts with the environment. The goal is to use such estimates to derive a policy \( \pi : S \mapsto A \)

\[
\pi(s) = \arg \max_a Q(s, a) \tag{2}
\]

The \( Q \) function is a state-action value-function that estimates the expected future rewards an agent would receive by taking a particular action from a particular state. Equation (4) defines how \( Q \)-values must be updated, considering a Temporal Difference (TD) target \( \delta \), which depends on the current reward \( r(s_t) \), the quality of the best action at the next state \( s_{t+1} \), a discount factor \( \gamma \leq 1 \) that decreases the value of future actions and a learning rate \( \alpha \) that defines the speed of convergence towards the observed \( Q \)-values.

\[
\begin{align*}
\delta(s_t, a_t) := & \ r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \quad \tag{3} \\
& s_{t+1} = s_t a_t + \alpha \delta(s_t, a_t) \quad \tag{4}
\end{align*}
\]

\( Q \)-learning has been extensively applied to many sequential decision problems in small-scale problems, where the state-action values \( Q(s, a) \) can be stored in a table. However, as the size of the state- and action spaces grow, function approximation is required to estimate the state-action values.

The Deep Q-Networks (DQNs) [27] is one example of successful use of NN as function approximators. The ability of NN to deal with large state spaces paved the way to the many recent successes of RL in large-scale environments. Although effective training of DQNs requires many technicalities [28], [29], its main result is straightforward: substitutes the tabular \( Q \) function by a parameterized NN \( Q_\theta \), resulting in a policy

\[
\pi_\theta(s_t) = \arg \max_a Q_\theta(s_t, a) \tag{5}
\]

Two important contributions of DQN for stabilizing the training of deep RL algorithms were the use of an experience buffer \( D = \{ (s_t, a_t, s_{t+1}, r_t) \} \), from which experiences \( e \sim D \) are sampled for training, and the use of an auxiliary target \( Q^\phi \) value function, whose parameters \( \theta^\prime \) are updated with a delay regarding \( Q \theta \). The resulting loss function can then be defined as

\[
\mathcal{L}_D(\theta) = \mathbb{E}_{e \sim D} \left[ (Q_\theta(s_t, a) - (r_t + \gamma \max_a Q_\theta^\prime(s_{t+1}, a)))^2 \right] \quad \tag{6}
\]

While the online network (with parameters \( \theta \)) is updated regularly through back-propagation on the gradients of the loss function, the target network (with parameters \( \theta^\prime \)) is periodically copied from the online network and never directly optimized. The use of both experience replay and the target network contributes to stabilizing the training.

The DQN proposal [27] was a significant step forward on the combination of reinforcement learning with deep learning. The synergy between Q-learning, function approximation, experience replay and target networks enabled agents to achieve human-level performance in many Atari games, and renewed the interest in deep reinforcement learning.

2) Continuous action spaces

However, in many scenarios, the interactions between agent and environment are better specified by actions in continuous actions spaces. A good example is congestion control, where the agent has the task of defining the packet transmission rate given observations of the network’s current state.

As we notice in Equation (5), the choice of the action with maximum \( Q_\theta(s, \cdot) \) becomes an expensive optimization problem in continuous actions spaces whose immediate solution is impractical. Fortunately, there are specialized RL algorithms that deal with such costs. As a representative example, we describe the Deep Deterministic Policy Gradient (DDPG) [30], which employs an actor-critic architecture combining both Q-learning and policy gradient techniques [26].

The actor is a deterministic policy \( \mu_\theta : S \mapsto A \) that, given a state, outputs actions in a continuous-space, while the critic is a parameterized Q-value function \( Q_\phi : S \times A \mapsto \mathbb{R} \) that assess the quality of such actions in terms of expected future rewards. Such functions are usually two NNs whose training are dependent on each other. Like the DQN, the DDPG also relies on an experience buffer \( D = \{ (s_t, a_t, s_{t+1}, r_t) \} \), and delayed target value functions \( Q^\phi \) and policies \( \mu^\phi \).

Training of the critic, with parameters \( \phi \), is based on TD learning and the minimum squared error, similarly to DQN. However, instead of computing the action that leads to maximum \( Q^\phi \), it uses the value for the action provided by the target policy \( a_{t+1} = \mu^\phi(s_{t+1}) \).

\[
\mathcal{L}_D(\phi) = \mathbb{E}_{e \sim D} \left[ (Q_\phi(s_t, a_t) - (r_t + \gamma Q^\phi(s_{t+1}, a_{t+1})))^2 \right] \tag{7}
\]

By fixing the parameters of the critic (\( \phi \)), the actor, with parameters \( \theta \), can be trained via the deterministic policy gradient algorithm, towards actions that maximize \( Q^\phi \).

\[
\theta = \arg \max_\theta \mathbb{E}_{e \sim D} \left[ Q_\theta(s_t, \mu_\theta(s_t)) \right] \tag{8}
\]

Differently from DQN, DDPG updates its target value-functions and policies via a procedure known as Polyak averaging that aggregates all previous versions of the NNs [30].

Despite the promising results shown by DDPG in multiple environments, it suffers from the overestimation bias of Q-values [32], which can induce the agent to converge towards bad quality local optima due to occasional peaks of rewards. The Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm addresses such shortcomings by introducing three features: (1) clipped double Q-Learning, (2) delayed policy updates (and target networks), and (3) target policy smoothing [35]. TD3 is the baseline RL agent used for the experiments in this paper.

**B. CONGESTION CONTROL**

Congestion control and avoidance (CC) deals, in general terms, with the reallocation of resources among competing...
nodes, with goals such as preventing congestion collapse, maintaining fairness and optimizing performance [34]. Many variations on this general setting exist, depending on aspects such as (a) whether cooperation of intermediate nodes is possible [35], (b) the specifics of traffic and environment [36], (c) whether the involved links have time-varying characteristics such as capacity and loss probability [37], (d) the contenders’ observations and actions [12], [14], among many others.

Research work in the congestion control field is still intensive even after thirty years of the development of the first congestion control algorithm [38]. Cubic TCP [39] is currently the default congestion control algorithm adopted by the Linux kernel, while Windows usually adopts Compound TCP [40]. In the latest kernel versions, Linux also allows the switch to Bottleneck Bandwidth and Round trip propagation time (BBR) congestion control [41]: BBR is being adopted by Google in Youtube servers and is under standardization by the Internet Engineering Task Force (IETF) [42].

As mentioned in other works [13], fronthaul congestion control research is currently in its early stages because most C-RAN implementations still adopt dedicated CPRI links [43]. But there has been recent contributions to congestion control literature in other network scenarios that provide useful insights and research questions for this work, and they will be discussed in sequence below.

From the point of view of fronthaul scenarios, among the state-of-the-art protocols it is important to highlight BBR and Timely [44] algorithms. Timely was implemented targeting datacenter deployments with stringent RTT requirements, in response to the complex protocols adopted for such scenarios (like Data Center TCP (DCTCP) [45]).

The Timely authors report similar results with less complexity, relying on accurate RTT measurements (other protocols require network support for explicit congestion notification, flow abstraction, etc). Strict delay and throughput requirements are observed in both datacenter and fronthaul scenarios, but there are still relevant discrepancies such as the topology (datacenter topologies are usually characterized as a Clos architecture [46], while current fronthaul scenarios are more like dumbbell and tree topologies).

Some works point to limitations with this approach. For example, the authors in [47] argue that the use of RTT measurements can make it too sensitive to congestion on the reverse path; other studies such as [48] indicate that delay-based protocols like Timely often have worse completion time of short flows, if compared with approaches based on Explicit Congestion Notification (ECN) signals.

BBR, on the other hand, is a hybrid protocol that theoretically estimates the available bandwidth and RTT, in theory running at Kleinrock’s optimal operating point. But recent works [49] demonstrated several fairness issues, especially when BBR shares resources with TCP Cubic flows. At the time of this writing, BBR v2 was under development to handle these issues, at the cost of additional complexity (e.g. inclusion of ECN signaling).

A traditional concern in the adoption of new and disruptive congestion control protocols is their coexistence with regular TCP flows, especially Cubic. Current fronthaul use cases are still a bit far from coexisting with other types of traffic and tend to focus on investigating protocols that can handle both microsecond-order delay requirements and very high throughput performance, guaranteeing fairness among flows. However, it is also becoming clear that we need more future-proof congestion control solutions given that 5G and Beyond 5G (B5G) scenarios include a huge variety of applications with even more stringent requirements. With this and the recent research trends towards the adoption of Machine Learning (ML) in communications, there are already several ML-based congestion control protocols being proposed, and in this work we focus on the challenge of real-time inference, since not all NN architectures can be directly deployed in network nodes due to computational complexity.

III. METHODOLOGY

This section defines the methods required to evaluate if policy distillation is a effective way of dealing with impetuous inference time of NNs (ms) in fast control loops (µs), such as those found in fronthaul networks. First, we describe the simulation environments where the TD3 agent is trained and the underlying learning problem, next we define the policy distillation process that converts the trained TD3 policy into decision trees.

A. NETWORK ENVIRONMENT

For the environment, a fronthaul simulation was developed on the Network Simulator 3 (NS-3), while the ns3-gym [50] open-source tool was employed to interface with RL agent. The fronthaul scenario was implemented as a UDP-based constant bit-rate communication between pairs of senders (DUs) and their receivers (RUs) sharing a common link in the dumbbell topology of Figure 2. Senders and receivers are connected to two switches through individual access links; and these switches exchange packets via the shared communication link. To isolate the dynamics we want to explore, we assume the access links have negligible packet losses and sufficient capacity, so that these impairments are only present in the shared link. To perform the congestion control on top of the UDP-based communication, agents are given control of the intersend time between sent packets, implementing a rate-based congestion control approach.

Typically, CC are event-based algorithms, in that they act upon the occurrence of a trigger event, for instance reception of an ACK or detection of a lost packet from a timeout. The RL agent, on the other hand, was implemented as a time-based algorithm. This decision was made to give the agent a better opportunity to observe the impact a previous action has had on the environment. Therefore, this observation period is also a simulation parameter that needs to be defined and ultimately also influences the overall performance of the agent [21], [25].

From an RL perspective, the proposed analysis required a scalable robust execution environment able to cope with the large number of simulations needed to train the agent. As
Such an execution environment was set up on an OpenStack cluster, employing the RAY framework\cite{51} as the main tool for managing the distributed computation and experiment lifecycles. As for the TD3 agent, we used the implementation available on RLlib\cite{52}.

Because the network is simplified in a dumbbell topology, the ratio between the total bandwidth required by all nodes and bottleneck size is a key parameter, and different bottleneck configurations were evaluated. Another important parameter is the duration of each simulation — in our case, each a simulation episode runs for 3 s during training.

B. LEARNING PROBLEM

While the setup fits a multi-agent scenario, i.e., senders must cooperate in order to maximize overall performance, the proposed approach was to accomplish such cooperation indirectly, where agents do not communicate among themselves and are not explicitly aware of how many senders are currently active on the network. Furthermore, each sender experience is collected locally and used to train a single policy that is shared by all agents. Therefore, all utilities and reward functions are calculated based on each agent’s local observations.

We define an observation space with four features, $S \subset \mathbb{R}^4$, the average RTT within the current time step ($\hat{RTT}_t$), the ratio of packet losses over the number of packets sent, the current intersend time value ($\Delta_t$) and the ratio between the average RTT and the minimum observed RTT. We chose those dimensions to provide the agent with estimates of its current RTTs, throughput, and losses, along with a reference regarding the minimum possible RTT.

Since the chosen agent (i.e., TD3) is capable of handling continuous action spaces, we defined a one-dimensional action space $A = [0.8, 1.5]$. The actions $a \in A$ are output by the agents’ policy and used to update the intersend time between packets. Therefore, at each time step, the agents use their local observations to estimate the global state of the network and decide how to update their transmission rate.

$$\Delta_t = \Delta_{t-1} / a$$

Finally, the rewards were defined based on three measurements from the current timestep, (a) the number of acknowledged packets, (b) the average round trip time, and (c) the number of packets lost. The rewards received at a timestep $t$ are a linear function of such components. However, to avoid issues with the scale of the components, they are first normalized to the same interval using a function $\eta : \mathbb{R} \times \mathbb{R}^2 \mapsto [0, 10]$. Since the reward function is not useful during the evaluation, the normalization function can rely on information that is not readily available after training, e.g., the bounds for number of acknowledged packets ($A$), RTTs ($R$), and packet losses ($L$).

$$r_t = \eta(\text{acks}_t, A) - \eta(\hat{RTT}_t, R) - \eta(\text{losses}_t, L)$$

Equation 10 aggregates three objectives in one, inducing the trained policies to increase the transmission rate until the point where they notice increases in RTT or packet losses. Figure 3 illustrates that behavior for scenarios with and without congestion. An optimal policy would discover the highest transmission rate that achieves fairness and high link utilization while keeping the RTTs close to the minimum, i.e., the peaks in the figure below.

$\text{Figure 3: The agents are rewarded for increasing their throughput but also penalized if the RTTs or packet losses increase.}$

The above formulation models congestion control as a decentralized decision-making problem in which the agents can only rely on local observations and rewards. Although that is a harsh environment for learning, due to partial observability and the inherent non-stationarity of the network state, it makes the scenario close to real-world deployments in which congestion control algorithms work in decentralized fashion.

C. POLICY DISTILLATION

Knowledge distillation refers to procedures that extract the knowledge from complex models to simpler ones. The hypothesis underlying such methods consists of two parts (1) large models, such as deep neural networks, can achieve higher accuracies than simpler models due to their ability
for learning meaningful representations, and (2) the learned representations can facilitate training of a simpler model without significant accuracy deterioration [53], [54].

Given the RL challenge of real-time inference, knowledge distillation is a promising method, seen that simpler models also require less time to output a result, i.e., smaller inference time. Therefore, if we are able to distill an RL policy into simpler model, we may decrease the inference time to reach a required target, e.g., sub-millisecond time scale. From now on, we refer as policy distillation the process of training a second policy (student) from data extracted from a first RL policy (teacher) [55], [56].

Given a teacher as a deterministic TD3 policy \( \mu_\theta : \mathcal{S} \rightarrow \mathcal{A} \), the first step of policy distillation is to collect experiences from \( \mu_\theta \) in the target environment. Such experiences are then stored as state-actions tuples in a dataset \( D = \{(s_i, a_i)\}_{i=1}^N \), where \( s_i \in \mathcal{S} \) are observations from the current state and \( a_i \in \mathcal{A} \) are the respective actions proposed by the policy, i.e., \( a_i = \mu_\theta(s_i) \).

From the data set \( D \), a student policy with a set of desired characteristics can be trained. Here, our focus is to decrease inference-time to the sub-millisecond time-scale, therefore, we chose a set of models that are known for their low inference cost: Decision Trees (DTs) and Gradient Boosting Trees (GBT). Models based on decision trees have the benefit of being easily converted to source code, which enables the resulting policies to be easily deployed as C/C++ code (see Figure 4).

FIGURE 4: The distillation process employed here produces two results: (1) a Python policy (DT) and (2) its respective version in C++.

The distilled policies are likely to show different performance and inference time depending on the data set size \( N \), the maximum allowed depth \( (d) \) for the decision trees and the number of boosting trees \( (b) \). Therefore, each distillation cycle must define values for all those parameters \( (N, d, b) \), where \( b \) only makes sense for GBTs. The experiments section show results for different settings of policy distillation and the trade-off between model complexity and inference-time.

IV. EXPERIMENTS AND RESULTS

This section describes experiments that evaluate how well distilled DTs compare to a trained TD3 agent, both in terms of performance (link utilization, packet delay, fairness) and inference-time. The experiments were set to assess the robustness of TD3 and DTs to network scenarios not seen during training and distillation settings. First, we describe the experiments’ settings used for training the TD3 agent. Second, we assess the performance and inference-time of multiple distilled policies. Finally, we provide a visual comparison of the policies.

A. TD3 TRAINING SETUP

Training of RL agents requires the setting of many parameters and hyperparameters for both, the environment (NS-3 simulations) and the RL algorithm (TD3 in this case).

Regarding the training environment, we considered a set of possible different network scenarios, each defined by the bottleneck capacity and the number of sender-receiver pairs. At every training episode, we choose random bottleneck capacities \( (C \in [1, 2000] \text{ Mbps}) \) and number of senders \( (p \in \{1, 2\}) \). By imposing such a variety of simulation environments during training, we expect to improve the resulting agent’s generalization ability. Such an approach is known in the literature as domain randomization [59]. Table 1 defines the domain for the main simulation parameters, but only \( C \) and \( p \) undergo domain randomization.

TABLE 1: Network simulation environment settings.

| Parameter               | Domain                          |
|-------------------------|---------------------------------|
| Bottleneck capacity     | \( 1 \text{Mbps} \leq C \leq 2 \text{Gbps} \) |
| Number of senders       | \( p \in \{1, 2\} \)            |
| Access links            | \( 20 \text{Mbps} \) (overprovisioned) |
| Switch queue size       | 100 packets                     |
| Simulation duration     | 3s                              |
| Timestep                | 1ms                             |

The TD3 agent specifies many other parameters that we can tune to enable effective training. Here we describe the main ones while referring to the appendix for a complete listing. Like any other DRL algorithm, the NN architecture and the learning rate have a significant impact on how effective the agent’s training is. TD3 employs a NN for actor and another for the critic, with possibly different learning rates. Table 2 shows the settings used for this paper.

TABLE 2: TD3 agent internal settings.

| Parameter               | Value                          |
|-------------------------|--------------------------------|
| Actor and critic architecture | \( 512 \times 16 \times 512 \) |
| Actor learning-rate     | \( 1 \times 10^{-5} \)         |
| Critic learning-rate    | \( 5 \times 10^{-5} \)         |
| Multi-step learning     | \( n = 5 \)                    |
| Discount factor         | \( \gamma = 0.999 \)           |
| Polyak averaging factor | \( \tau = 5 \times 10^{-5} \)   |

Other parameters that also impact training are those related to Q-learning. We observed that encouraging the agent to look further into the future favored cooperation, so we defined \( \gamma = 0.999 \) and used multi-step learning with \( n = 5 \).

B. PERFORMANCE EVALUATION

The performance evaluation consists of comparing the TD3 policy and the distilled decision trees. We evaluate the
policies in a wide range of network scenarios, which expose the policies to different network conditions. TD3 policies actuate every 1 ms, while DT policies actuate at every ACK. Later on, we aggregate all the results to quantify the overall performance. Table 3 defines values to four parameters that specify the possible network simulation scenarios.

| Parameter             | Value                |
|-----------------------|----------------------|
| Trees depth           | \(d = 6, 10, 15\)    |
| Bottleneck capacity (Gbps) | \(C = 0.5, 1, 1.5\) |
| Number of trees       | \(T = 0, 1, 5, 10, 20, 40\) |
| Number of senders     | \(p = 1, 2, 8, 16, 20\) |
| Number of episodes distilled | \(N = 100, 500, 1000\) |

The different bottleneck capacities induce network scenarios with different RTT ranges, while the number of senders impacts the network dynamics. An optimal agent would maintain high link utilization (close to 100% of the bottleneck capacity), average RTTs close to the minimum, and zero packet losses while guaranteeing fair bandwidth shares for each flow. It is also desirable that the trained policies generalize well to a superset of the scenarios seen during training. For that reason, the results here contain simulations with as many as 20 sender-receiver pairs, much more than those observed during training.

Each scenario induces different ranges for the performance measurements. Therefore, the results are first normalized and only then aggregated. In this sense, whenever we mention link utilization close to one or RTTs close to zero, we are referring to a link utilization close to the maximum and RTTs close to the minimum. Such an approach allows us to summarize all results in a few figures (see Figure 5).

Figure 5a shows the aggregated results for all policies in all network scenarios evaluated. We have link utilization, RTTs, packet losses, and the Jain fairness index from left to right. TD3 shows high link utilization, but surprisingly the distilled DTs achieved an even higher average at the cost of increased variance. Regarding RTTs, the DTs improved as depth increased, maintaining smaller averages than TD3. Regarding packet losses, DTs were not so effective, with a small number of packet losses occurring. Finally, the DTs improved fairness as their depth increased, achieving the same performance as TD3. The overall results validated the proposal of policy distillation from the perspective of performance. However, they do not show how the policies perform in specific network scenarios. To reach that, we aggregate the results by different aspects and analyze them separately.

Figure 5b, split the results by the bottleneck capacity \(C = 500, 1000, 1500\) Mbps and aggregates the performance measurements within each of those classes. Such an approach provides evidence for how the policies perform depending on the bottleneck capacity. The performance profile follows the overall pattern, with the DTs reaching higher link utilization but with higher variance and similar RTTs and fairness.

However, regarding packet losses, we can now see that they are occurring only for the tightest bottlenecks, i.e., 500Mbps. Figure 5c shows the results from the perspective of the number of trees used by the distillation process. The TD3 does not use trees and it is indicated by \(T = 0\). On the other hand, the simple decision trees use only one tree, i.e., \(T = 1\). For all values \(T \geq 2\), the Gradient Boosting Trees are being employed. We see clear improvements when the number of trees increases until \(T = 20\), with benefits for link utilization, RTTs, and fairness. However, for \(T = 40\), the benefits are not clear, since the improvements in link utilization came at the cost of slightly higher RTTs (in comparison to \(T = 20\)), worse fairness (e.g., DT10 with \(T = 40\)), and increased packet losses.

Figure 5d aggregate the results by the data set size used for the policy distillation process \(N = 100, 500, 1000\). Each data set is composed of experiences collected from several episodes. Each episode defines a randomized network scenario with different bottleneck capacity \(C\) and many sender-receiver pairs \(1 \leq p \leq 8\). Therefore, this figure shows the impact of increasing the data set size on the distilled policies. The link utilization, RTTs, and fairness of the DTs improved as we increased \(N\) from 100 to 500, but further increases did not seem to provide many benefits. It is also worth noting that packet losses occurred for \(N = 100, 1000\).

Figure 5e shows the results aggregated by the number of sender-receiver pairs \(p\). That allows us to observe the performance of the distilled policies with different environment dynamics. The first important aspect to notice is the low link utilization for the DTs policies in scenarios with \(p = 1, 2\). Since the TD3 policy does not show the same behavior, we conclude the distillation process causes this issue. Most likely, the data sets used to learn the DTs are biased towards scenarios with a larger number of pairs. In retrospect, scenarios with a larger number of senders produce more experiences than those with fewer senders. Therefore, the data set contains more experiences from such scenarios, reflecting on the DTs. Despite most results being close to the maximum fairness index, we see a small deterioration of the average. Finally, we identify packet losses occurring only for scenarios with \(p = 16, 20\).

As we mentioned, some of the distilled DTs showed a deteriorated performance compared to TD3. However, from Figure 5f we conclude that only occurs for network scenarios out of the training distribution, i.e., \(p = 16, 20\). That shows that the policy distillation process employed here produces DTs that do not generalize very well. Therefore, to better analyze the relation between DTs’ depth and their performance, we consider now only results from network scenarios from the training distribution, i.e., \(p \leq 8\). Figure 6 summarizes such results, where we see a clear benefit of having deeper DTs, with improvements on link utilization, RTTs, and fairness without any packet losses.

All these results confirm that it is possible to distill a TD3 policy into DTs maintaining the efficacy of the original policy to some extent. We investigate now if the distilled policies...
(a) Overall results for all different network scenarios.

(b) Aggregated by bottleneck capacity: $C = 500, 1000, 1500$ Mbps.

(c) Aggregated by number of trees: $T = 0$ (TD3), 1 (Decision tree), 2 or more (Gradient Boosting Trees).

(d) Aggregated by distillation dataset size: $N = 100, 500, 1000$ episodes collected.

(e) Aggregated by number of sender-receiver pairs: $p = 1, 2, 8, 16, 20$.

FIGURE 5: The boxplots show the average performances of all congestion control policies in all network scenarios. We denote the decision trees as DT, followed by a number indicating their maximum depth. While 5a shows the overall results, the other figures show results aggregated by: 5b the bottleneck capacity, 5c the number of trees, 5d the size of the dataset used by the distillation process, and 5e the number of pairs.
meet the inference-time requirements of fronthaul networks.

C. INFEREN-C-TIME EVALUATION

Inference on NNs is a forward pass from the input until the last layer. That consists of a series of matrix multiplications whose complexity depends on the number of layers and their sizes. Although inference-time can be highly optimized by using specialized hardware, such as Graphics Processing Units (GPUs), it is not realistic to assume the availability of such hardware in the network nodes. DTs inference complexity, on the other hand, only depends on the depth of the trees and the number of trees employed, which allows them much smaller inference-time. This section quantifies the benefits of policy distillation regarding inference-time.

To estimate the average inference-time of TD3 and the distilled DTs we performed $1 \times 10^5$ measurements for each policy. We also consider two versions for each policy, the original Python objects and the equivalent C++ code. The TD3 Python policy was directly obtained from RLlib, while we obtained its C++ counterpart by using the CppFlow library. The Sklearn DTs were exported to C++ by adapting an external code, while the GBTs could be directly exported using support already provided by the CatBoost library.

Figure 7 shows the distribution of these results for each policy in microseconds (µs) and aggregated by (1) the number of trees and (2) trees’ depth. While we see only a slight improvement on inference-time by using CppFlow on the TD3 policy model, all the C++ DTs showed significant improvements in comparison to their Python counterpart. Overall, that means reductions of the order of 99%, i.e., from 1ms to 1µs.

These results imply policy distillation as an effective way of learning policies that meet stringent inference-time requirements without considerable performance deterioration. Additionally, we observe that inference-time grows faster with $T$ (number of trees) than with $d$ (depth). The particular choice of $(T, d)$ depends on the application scenario requirements. For example, in the case of fronthaul networks, a proper selection produces policies whose inference-time is much smaller than the minimum RTTs ($< 1$ ms). These policies can therefore adapt the transmission rate very frequently, e.g., at every ACK received, which avoids issues with delayed actuation already reported in the literature [21], [25].

D. POLICY VISUALIZATION

The results we have shown so far summarize the average performance and inference-time of all policies evaluated. This section shows the behavior of such policies during the simulations and compares a TD3 policy with its distilled counterparts. For conciseness, we use DTs with $T = 1$ as a reference since they provide a good trade-off of inference-time and performance.

Figure 8 exemplifies the TD3 behavior and the respective distilled policies, with $d = 5, 10, 15$, on a network simulation with bottleneck capacity of 1Gbps and eight sender-receiver pairs. Although TD3 shows the cleanest behavior, with high link utilization, small RTTs and high Jain fairness index, the
FIGURE 8: The performances in throughput (Mbps), losses, RTT (us), and fairness. Network scenarios with $p = 8$ and $C = 1$Gbps. The maximum link utilization is represented by the dashed black line, the achieved link utilization is gray.

distilled DTs are not significantly behind.

An alternative to performance comparison is to compare the policies directly as heatmaps, i.e., a mapping from the observation space to the individual actions. However, the observation space used in the experiments comprises four dimensions, and we cannot directly represent them as a 2D plane. To circumvent that limitation, we consider only the two most representative dimensions, according to feature-importance information provided by the catBoost library (i.e., intersend and RTT ratio), and average out the actions over the remaining dimensions. Additionally, we only consider the hyperplane in which packet losses equals zero.

Figure 9 shows the results for such a procedure, where the colors indicate the action values ($a \in [0.8, 1.5]$, from red to blue). We clearly see the DTs policies becoming more similar to the TD3 policy as their depth increases from $d = 5$ to $d = 15$. Such increasing similarity explains the performance results seen earlier and shows that there is still a significant difference between TD3 and DT15. Indeed, the contour lines of TD3 are smoother than those of the DTs, a result of NNs increased ability to deal with non-linear decision spaces.

FIGURE 9: Visualization of TD3 and DT policies. As depth increases, the distilled decision trees become more similar to the TD3 policy.

To summarize the results, Figure 10 shows the bivariate distribution of link utilization and RTT aggregated over all experiments for two representative policies, the original TD3 and the resulting DT with $T = 1$ and $d = 15$ (DT15). Since the results are scaled to the range $[0, 1]$, an optimal congestion control policy would reach link utilization close to one and RTTs close to zero. The distributions show that both policies reach overall performance close to the optimum, with DT15 reaching a more concentrated distribution regarding RTTs.

V. CONCLUSIONS

The evolution of 5G with C-RAN architectures imposes stringent demands on transport networks. On the one hand, we can achieve a cost-efficient solution to deal with the high throughput requirements by relying on statistical multiplexing enabled by packetized network deployments. On the other hand, the use of such non-dedicated links allows congestion to occur due to, for example, aggressive radio schedulers. Classical congestion control algorithms are likely not to perform well in such scenarios.

Recent efforts reported in the literature have shown Reinforcement Learning (RL) as an effective alternative to such cases in which classical congestion control algorithms do not succeed. However, RL policies based on Neural Networks (NNs) come with a higher inference cost that the network hardware infrastructure might not appropriately support. That might result in inference-time being higher than the RTTs, rendering such RL policies unable to update the transmission
which is worse than what was shown by the TD3 policy, and enabling them in practice. The main drawbacks of the technology management.

In high-speed control loops, such as congestion control for reducing inference-time by maintaining similar performance to TD3 in most scenarios while are translatable to C++ code. We showed that such C++ DTs are Decision Trees (DTs) that mimic its behavior. DT models that have lower inference-time complexity than NNs, and they are translatable to C++ code. We showed that such C++ DTs maintain similar performance to TD3 in most scenarios while reducing inference-time by 99%, i.e., from millisecond to sub-microsecond time-scale. That is enough for enabling DT in high-speed control loops, such as congestion control for fronthaul networks.

The results show that the method proposed is a viable alternative for reducing the inference-time of RL policies, enabling them in practice. The main drawbacks of the technique regard (1) the generalization ability of the distilled DTs, which is worse than what was shown by the TD3 policy, and (2) the diminished flexibility, in the sense that distilled DTs make the management of the policies life-cycle more complex. For future research, we consider that (1) could be tackled by more sophisticated strategies of collecting experiences from TD3, while we see (2) as natural trade-off between real-time inference requirements versus an easier life-cycle management.

REFERENCES

[1] M. Jaber, M. A. Imran, R. Tafazoli, and A. Tukmanov, “5G backhaul challenges and emerging research directions: A survey,” IEEE Access, vol. 4, pp. 1743–1766, 2016.

[2] H. Holma and A. Toskala, LTE advanced: 3GPP Solution for IMT-Advanced. John Wiley & Sons, 2012.

[3] D. Chitimalla, K. Kondepu, L. Valcarcegui, M. Tornatore, and B. Mukherjee, “5G fronthaul–latency and jitter studies of cpri over ethernet,” Journal of Optical Communications and Networking, vol. 9, no. 2, pp. 172–182, 2017.

[4] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, “End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks,” Computer Networks, vol. 186, p. 107692, 2021.

[5] L. Zhang, Y. Cui, M. Wang, Z. Yang, and Y. Jiang, “Machine learning for internet congestion control: Techniques and challenges,” IEEE Internet Computing, vol. 23, no. 5, pp. 59–64, 2019.

[6] 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. PMLR, 2019, pp. 3050–3059.

[7] T. Zhang and S. Mao, “Machine learning for end-to-end congestion control,” IEEE Communications Magazine, vol. 58, no. 6, pp. 52–57, 2020.

[8] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, and M. Xu, “When machine learning meets congestion control: A survey and comparison,” arXiv preprint arXiv:2010.11397, 2020.

[9] W. Wei, H. Gu, and B. Li, “Congestion control: A renaissance with machine learning,” IEEE Network, pp. 1–8, 2021.

[10] A. Sivaraman, K. Weinstein, P. Thaker, and H. Balakrishnan, “An experimental study of the learnability of congestion control,” in ACM SIGCOMM Computer Communication Review, vol. 44, no. 4. ACM, 2014, pp. 479–490.

[11] K. Weinstein and H. Balakrishnan, “Tcp ex machina: Computer-generated congestion control,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 123–134, Aug 2013.

[12] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, “Qtcp: Adaptive congestion control with reinforcement learning,” IEEE Transactions on Network Science and Engineering, vol. 6, no. 3, pp. 445–458, 2019.

[13] Y. Kong, H. Zang, and X. Ma, “Improving tcp congestion control with machine intelligence,” in Proceedings of the 2018 Workshop on Network Meets AI & ML, ser. NetAI'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 60–66. [Online]. Available: https://doi.org/10.1145/3295553.3324300

[14] K. Xiao, S. Mao, and J. K. Tugnait, “Tcp-drinc: Smart congestion control based on deep reinforcement learning,” IEEE Access, vol. 7, pp. 11 892–11 904, 2019.

[15] M. Bachl, T. Zseby, and J. Fabini, “Rax: Deep reinforcement learning for congestion control,” in ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, 2019, pp. 1–6.

[16] L. Liu and H. Xu, “Tyrus: Phy-aided adaptive agent congestion control for cellular networks,” in Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, ser. SIGCOMM Posters and Demos ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 45–47. [Online]. Available: https://doi.org/10.1145/3342280.3342302

[17] V. Sivakumar, T. Rocktäschel, A. H. Miller, H. Küttler, N. Nardelli, M. Rabbat, J. Pineau, and S. Riedel, “Mfsvf-rl: An asynchronous rl framework for congestion control with delayed actions,” 2019.

[18] I. Nascimento, R. Souza, S. Lins, A. Silva, and A. Klautau, “Deep reinforcement learning applied to congestion control in fronthaul networks,” in 2019 IEEE Latin-American Conference on Communications (LATINCOM), 2019, pp. 1–6.

[19] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, “Smartcc: A reinforcement learning approach for multipath tcp congestion control in heterogeneous networks,” IEEE Journal on Selected Areas in Communications, vol. 37, no. 11, pp. 2621–2633, 2019.

[20] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, “Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning,” IEEE Journal on Selected Areas in Communications, vol. 37, no. 6, pp. 1325–1336, 2019.

[21] S. Abbasloo, C.-Y. Yen, and H. J. Chao, “Classic meets modern: A pragmatic learning-based congestion control for the internet,” in Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, ser. SIGCOMM ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 632–647. [Online]. Available: https://doi.org/10.1145/3387513.3408892

[22] S. Emaru, B. Li, and Y. Chen, “Eagle: Refining congestion control by learning from the experts,” in IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 676–685.

[23] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning,” 2019.

[24] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, “An empirical investigation of the challenges of real-world reinforcement learning,” 2021.

[25] L. Zhang, K. Zhu, J. Pan, H. Shi, Y. Jiang, and Y. Cui, “Reinforcement learning based congestion control in a real environment,” in 2020 29th
International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020, pp. 1–9.

[26] R. S. Sutton and A. G. Barto, Introduction to reinforcement learning. MIT Press Cambridge, 1998, vol. 135.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, p. 529, 2015.

[28] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, “Deep reinforcement learning and the deadly triad,” CoRR, vol. abs/1206.0268, 2012.

[29] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” CoRR, vol. abs/1710.02298, 2017.

[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,” 2015.

[31] 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. JMLR.org, 2014, p. 1–387–1–395.

[32] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, ser. AAAI’16. AAAI Press, pp. 2094–2100.

[33] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing approximation error in actor-critic methods,” 2018.

[34] S. Floyd, “Congestion Control Principles,” RFC 2914, Sep. 2000.

[35] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168, Sep. 2001.

[36] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, and G. Judd, “Data Center TCP (DCTCP): TCP Congestion Control for Data Centers,” RFC 8257, Oct. 2017.

[37] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, P. B. Godfrey, and S. Floyd, “Pcc vivace: Online-learning congestion control,” in Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI’18. USA: USENIX Association, 2018, p. 343–356.

[38] J. Postel et al., “Transmission control protocol,” 1981.

[39] 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.

[40] K. Tan, J. Song, Q. Zhang, and M. Sritharan, “A compound tcp approach for high-speed and long distance networks,” in Proceedings-IEEE INFOCOM, 2006.

[41] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “Btr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time,” Queue, vol. 14, no. 5, pp. 20–53, 2016.

[42] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Veness, J. Jacobson, Y. M. H. A. R. Patel, S. Bhatia, Y. Wang, D. Wetherall, and D. Zats, “Timely: Rtt-based congestion control for the datacenter,” ACM SIGCOMM Computer Communication Review, vol. 45, no. 4, pp. 537–550, 2015.

[43] M. Alizadeh, A. Greenberg, D. A. Matzi, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sritharan, “Data center tcp (dctcp),” in Proceedings of the ACM SIGCOMM 2010 Conference, 2010, pp. 63–74.

[44] J. Cheng and K.-I. Grinnemo, Telco Distributed DC with Transport Protocol Enhancement for 5G Mobile Networks: A Survey. Karlstad universitet, 2017.

[45] Y. Zhi, H. Eran, D. Firestone, C. Guo, M. Lipshyten, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, “Congestion control for large-scale rdma deployments,” ACM SIGCOMM Computer Communication Review, vol. 45, no. 4, pp. 523–536, 2015.

[46] Y. Zhu, M. Ghobadi, V. Misu, and J. Padhye, “Ecn or delay: Lessons learnt from analysis of dccq and timely,” in Proceedings of the 12th International Conference on emerging Networking Experiments and Technologies, 2016, pp. 313–327.

[47] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda, and S. Yamaguchi, “Tcp fairness among modern tcp congestion control algorithms including tcp bbr,” in IEEE 8th International Conference on Cloud Networking (CloudNet), 2018, pp. 1–4.

[48] P. Gavlowicz and A. Zubow, “ns3 gym: Opening openai gym for network-working research,” CoRR, vol. abs/1810.03943, 2018.

[49] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, M. Eli-bol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, “Ray: A distributed framework for emerging ai applications,” in Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI’18. USA: USENIX Association, 2018, p. 561–577.

[50] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, “Rllib: Abstractions for distributed reinforcement learning,” 2017.

[51] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, “Improved knowledge distillation via teacher assistant,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 04, 2020, pp. 5191–5198.

[52] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in NIPS Deep Learning and Representation Learning Workshop, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[53] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, “Policy distillation,” 2016.

[54] L. Wang and K. J. Yoon, “Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2021.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Edouard Duchesnay, “Scikit-learn: Machine learning in python,” Journal of Machine Learning Research, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[56] L. Prokhorovkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: Unbiased boosting with categorical features,” in Proceedings of the 32nd International Conference on Neural Information Processing Systems, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 6639–6649.

[57] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 23–30.

JEAN P. MARTINS received his Ph.D on Computer Science from the ICMC at the University of São Paulo (2015) and his masters from the Federal University of Goiás (2010). Since 2018 he works as an Experienced Researcher in Artificial Intelligence at Ericsson Research. His main research interests intersect combinatorial optimization, evolutionary algorithms, multi-agent systems and deep reinforcement learning.

IGOR ALMEIDA received in 2010 a B.Sc. degree in computer engineering, and in 2013 an M.Sc. degree in electrical engineering, both from the Federal University of Pará (UFFPA), Belém, Brazil, where he is currently pursuing a Ph.D. degree in electrical engineering. He has been with Ericsson Research since 2016, involved with topics such as 5G, fronthaul networking, synchronization and wireless communications.
RICARDO SOUZA received his B.Sc. degree in electrical engineering from the Federal University of Campina Grande (UFCG) in 2008 and in 2011 his M.Sc. in computer engineering from University of Campinas (Unicamp) where he is currently pursuing a Ph.D. degree. Since 2016 he works as an Artificial Intelligence Researcher at Ericsson Research Brazil. His research interests include distributed systems, multi-agent systems, deep reinforcement learning and networked robotics.

SILVIA LINS is an Experienced Researcher from the Networks area at Ericsson Research Brazil. She received her B. Sc. degree in Computer Engineering from the Federal University of Para (UFPA) in 2011 and her M.S. on Electrical Engineering in 2013 from the same university, where she is also doing her Ph.D. Currently her work focuses on machine learning for transport and mobile networks optimization.

**APPENDIX. TD3 PARAMETERS ON RLlib**

| TD3 specific | Value       |
|--------------|-------------|
| twin_q       | True        |
| policy_delay | 2           |
| smooth_target_policy | True       |
| target_noise | 0.05        |
| target_noise_clip | 0.1         |

**Exploration settings**

| type            | Gaussian    |
| random_timesteps | 1000000     |
| stddev          | 0.1         |
| initial_scale   | 1.0         |
| final_scale     | 1.0         |

**Experience buffer**

| buffer_size     | 2000000     |
| prioritized replay | True      |
| prioritized replay_alpha | 0.2       |
| prioritized replay_beta    | 0.2        |
| beta_annealing_timesteps   | 2 x 10^7   |
| final_prioritized replay_beta | 1 x 10^-6 |

***