problems, especially robotics [7]. We describe the design and architecture of MARLIN and discuss how we trained the agent, the hyperparameters used, and the main assumptions and choices we made during its design phase, comparing our work with other RL-based approaches to CC. Furthermore, we discuss the future directions of this research, of which this paper is just the first step. Finally, we present preliminary experimental results obtained in a real networking scenario, where we investigate the current performance of MARLIN and compare it to the Transmission Control Protocol (TCP) in handling a file transfer task.

II. Designing a Reinforcement Learning agent for Congestion Control

MARLIN is a RL agent for CC, trained in a real network by using continuous actions to update the CWND. For the purpose of this research, we did not consider fairness towards competing MARLIN flows and/or other protocols as one of the optimization objectives of MARLIN, which we leave as a future research question.

Within MARLIN, learning is based on SAC [11], which trains a stochastic, off-policy, entropy-regularized agent. The agent interfaces with a custom transport protocol, presented in Section III, via Remote Procedure Calls (RPCs) in a non-blocking, bi-directional fashion, as shown in Figure 1. The frequency of action taking depends on the latest Smoothed Round-Trip Time (SRTT) estimated for the path. The reward function is shaped as a strictly negative reward to intrinsically encode the principle that *every step taken to transfer the information is a step too much*.

Although it is generally desirable that the agent completes its tasks as quickly as possible, the RL setting proposed here does not aim for terminal states. It is inconvenient for the agent to overfit its trajectories on a fixed amount of bytes to transfer, or a fixed episode length, such that it will try to find a terminal state at any cost in a certain number of timesteps. Instead, we aim at designing RL agents able to work on "crowded" links and achieve high channel utilization independently from the volume of data transferred during training. For such reason, training is based on infinite-horizon tasks, never encountering a terminal state and following the Partial Episode Bootstrapping (PEB) principles for infinite-horizon training presented in [12].

A. Congestion Control as Markov Decision Process

Traditional CC algorithms implemented within transport protocols such as TCP take actions, e.g., updating their CWND, in response to changes in the environment they sense with the goal of achieving maximum throughput and minimum congestion. Changes may include, for example, packet loss, duplicate packets, or variations in the measured Round-Trip Time (RTT). Such sequential decision making setting can be naturally posed as a Markov Decision Process (MDP) [13].

The MDP framework formalizes with $\langle S, A, p, R \rangle$ the sequential agent-environment interaction as the exchange of three different signals at every time step t of the process. The

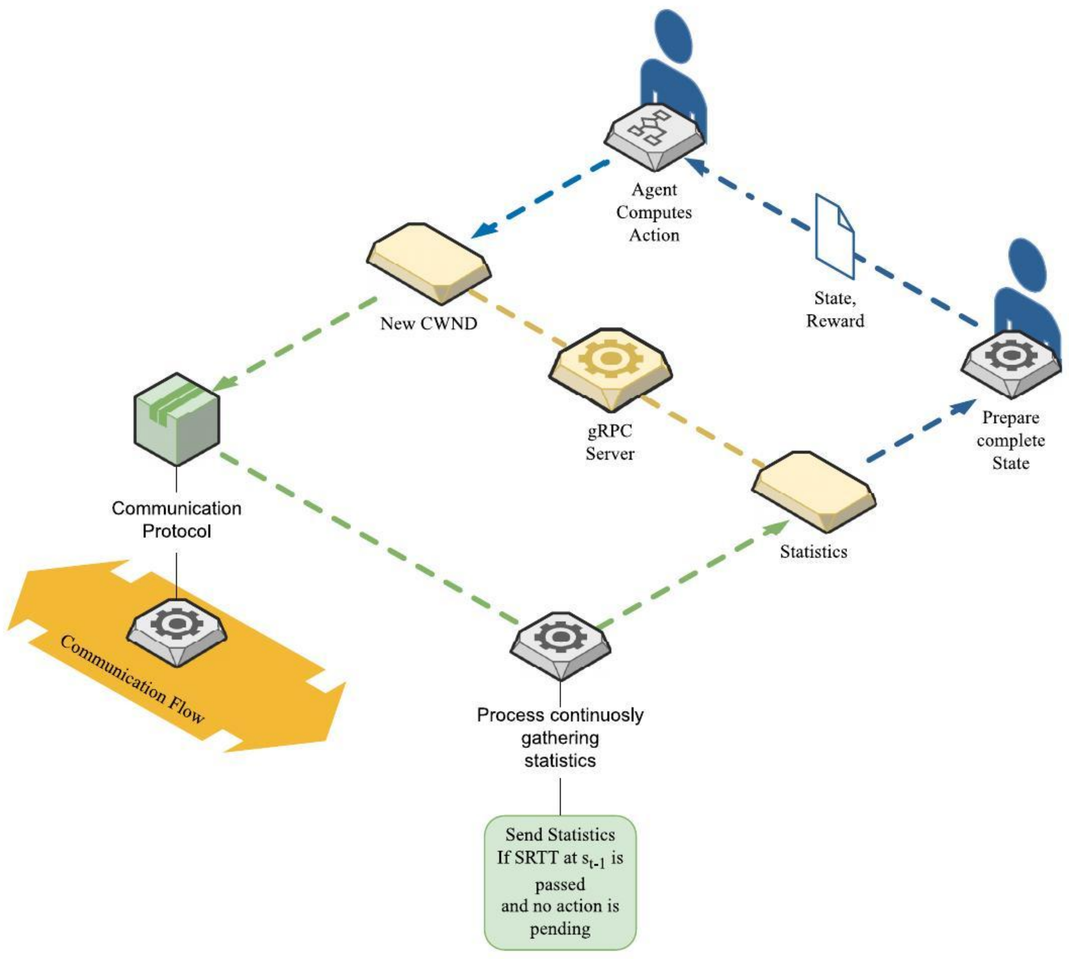

Figure 1: Agent-Protocol interface. Communication protocol and agent communicate through a gRPC server.

RL *"learning from interaction"* fundamentals [14] stem from this abstraction: the *action* $a_t \in A(s_t)$ taken by the agent at a certain step t after sensing the *state* $s_t \in S$ leads to a consequent response of the environment as a *reward* signal $r_{t+1} \in R \subset \mathbb{R}$ and transitions to a new state s_{t+1}. This process follows the environment dynamics described by the probability function $p(s_{t+1}, r_{t+1}|s_t, a_t) : S \times R \times S \times A \longrightarrow [0, 1]$. Such formulation of the transition dynamics also defines the *Markov property*, which requires the state to include all the relevant information about past interactions that would impact the future, as each possible values of $r_t \in R$ and $s_t \in S$ depend solely on the previous state and action s_{t-1}, a_{t-1}.

Communication networks are a subtle environment for a RL agent. Packets in modern networks are often transmitted a few milliseconds apart, sometimes microseconds. At the same time, running Stochastic Gradient Descent (SGD) to update the model during training, or even simply computing a new action, can take up a significantly greater amount of time. As a consequence, the environment might have already changed significantly by the time the action is taken, potentially making it obsolete or, even, wrong. This has been identified as one of the main reasons why it is hard to maintain the performance achieved in a simulated environment after moving to a real-world deployment [15].

Furthermore, actions do not instantly affect the agent's perception, as the effects of changing the CWND require some time before they have an impact on the network and that impact is propagated back to the source. As new packets are transmitted at a different rate after an action has been taken, they first need to reach their destination, then the receiver needs to generate and transmit a new ACKnowledge (ACK) message for those packets, and finally that ACK needs to arrive back at the source before any information about the impact of