The term $penalties$ depends on the difference between the current RTT and the minimum EMA RTT and is defined as follows:

$$penalties = \begin{cases} \alpha \frac{rtt_{diff}}{rtt_{min}^{ema}}, & \text{if } \frac{rtt_{diff}}{rtt_{min}^{ema}} < 1 \\ 0.99, & \text{otherwise} \end{cases} \quad (3)$$

In Eq. 3, α depends on the magnitude of the difference between rtt_{diff} and rtt_{min}^{ema} and it is defined as follows:

$$\alpha = \begin{cases} 1, & \text{if } |\frac{rtt_{diff}}{rtt_{min}^{ema}}| > 0.6 \\ 0.5, & \text{if } 0.1 < |\frac{rtt_{diff}}{rtt_{min}^{ema}}| \leq 0.6 \\ 0.3, & \text{if } 0.05 < |\frac{rtt_{diff}}{rtt_{min}^{ema}}| \leq 0.1 \\ 0.1, & \text{otherwise} \end{cases} \quad (4)$$

It is worth noticing that, in case $rtt_{diff} < 0$, the $penalties$ term becomes negative, thus rewarding the agent when RTT improvements are detected.

4) RL Algorithm: MARLIN adopts SAC [11], an off-policy actor-critic algorithm based on the maximum entropy RL framework. SAC augments the maximum reward objective, foundation of RL settings, with an entropy maximization term. Such term acts as a trade-off between exploration and exploitation, so that the agent aims to maximize its return while also acting as random as possible. In circumstances where multiple actions seem equally attractive, i.e. in the case of equal or close Q-Values, the learned policy is encouraged to assign equal probability mass to those actions.

In practice, the effects of the entropy, or temperature, term prompt the agent to discard unsuitable trajectories in favor of more promising ones, as well as to improve the learning speed. The entropy term can be either fixed or optimized/learned as further steps are taken. However, the optimal entropy coefficient varies depending on a series of factors, such as the nature of the task or even the current policy. As a consequence, it is usually considered good practice to avoid fixed values, preferring instead to update the term at the same time actor, critic, and the target networks are optimized [7].

D. Future Directions

The work described above is the first step of a multi-year research project. We have already identified future steps that follow naturally from our present work. While we proceed describing them sequentially, in reality these steps are much intertwined and we will likely addressed them in subsequent iterations.

We think that MARLIN would benefit from a more expressive reward function. We envision problem and reward formulations that truncate unpromising trajectories that have moved too distant from the optimal. We believe this could significantly speed up the solution convergence. Furthermore, we suspect that MARLIN's current state might present redundant information as well as features of low relevance to the problem. To address this, we plan to investigate smaller and more refined state representations, with the double goal of lowering complexity and improving convergence. We plan to train this new agent in diversified networking scenarios, which can capture different traffic patterns and network technologies, to assess the degree of generalizability. Finally, a thorough, automated hyperparameter tuning would further enhance MARLIN's performance and complete a first cycle of improvements.

Hyperparameter	Value
Training steps	1×10^6
History length	10
Training episode length	200
Learning rate	3×10^{-4}
Buffer size	5×10^5
Warm-up (learning starts)	1×10^4 steps
Batch size	512
Tau	5×10^{-3}
Gamma	0.99
Training Frequency	$1/episode$
Gradient Steps	-1 (same as episode length)
Entropy regularization coefficient	"auto" (Learned)
MLP policy hidden layers	$[400, 300]$

Table II: Hyperparameters used in MARLIN.

Further advancements will require an evolution in the agent's design congruent with specific problems that afflict CC. The literature has shown that ML models can accurately distinguish between packet loss attributable to congestion or channel errors. We plan to integrate a similar classifier within MARLIN and investigate the feasibility of an analogous approach to identify variations in end-to-end latency that are caused by changes in the path to destination. The fundamental building block of CC algorithms is *how and when* to change the CWND size. MARLIN currently actively controls the "how", leaving the "when" to a heuristic. We will investigate learning-based approaches to include such decision factor into MARLIN, with the goal of turning it into a more comprehensive and reactive system, able to make rational decisions at its own *tempo*.

Up until now, we shaped the problem of CC from the perspective of a single RL agent. Nonetheless, the need for CC algorithms in transport protocols originated from the lack of coordination in a multi-agent system, where single entities were acting in a completely self-centered manner. Therefore, we expect that the next step ahead in learning-based CC will come from the application of advancing Multi-Agent Reinforcement Learning (MARL) algorithms that can optimize cooperative and/or competitive agent behavior.

III. The Mockets Transport Protocol

One of the main design choices we faced was the transport protocol we would use to train and evaluate MARLIN. The decision fell on a custom transport protocol because it is much simpler to integrate with MARLIN than TCP. Additionally, a custom protocol enables greater flexibility in terms of retrieving the information required to encode the agent state.

For this study, we integrated MARLIN with Mockets, a message-based communication middleware implemented on top of UDP, following a school of thought similar to the one