	Feature	Description		Statistic
1	Current CWND	Current CWND	1	Last
2	KBs Sent	Amount of KB sent *	2	Mean
3	New KBs sent	Amount of KB acked *	3	STD
4	Acked KBs	Amount of KB acked *	4	Min
5	Packets sent	Packets sent *	5	Max
6	Retransmissions	Number of packets retransmitted *	6	EMA
7	Instantaneous Throughput	Throughput *	7	Difference from Previous
8	Instantaneous Goodput	Goodput *		
9	Unacked KBs	Amount of KBs in flight		
10	Last RTT	Last RTT detected *		
12	Min RTT	Min RTT *		
12	Max RTT	Max RTT *		
13	SRTT	Smoothed RTT *		
14	VAR RTT	RTT variance *		
		* During the last RTT timeframe		

Table I: Features composing the state of the environment. The horizon of the observation is also augmented with the previous 10 observations.

LIN takes care of delivering observations and reward pairs to the agent and actions back to the protocol.

At a given step t, the agent receives network statistics from the transport protocol via the gRPC middleware. Statistics are then processed and stacked with the previous 10 observations to form the state s_t. The agent can take actions that will increase, maintain constant, or decrease the transport protocol CWND by a chosen factor. Ideally, the agent should learn to maximize the volume of data transmitted in the minimum amount of time, while being watchful of growing queueing delays, which would manifest with an increase in the measured RTT.

MARLIN is implemented on top of the RL Baselines3 Zoo (RL-Zoo3) [20] framework, which follows best practices for using Stable-Baselines3 (SB3) [21], a PyTorch [22]-based library that implements state-of-the-art RL algorithms following the OpenAI gym interface [23].

1) State: Table I describes the state encoded in MARLIN. 14 features are gathered during the time frame that follows the action taken at step $t-1$. MARLIN then augments the state space with 7 statistics, i.e., last, mean, standard deviation, minimum, maximum, Exponential Moving Average (EMA), and difference from the previous state s_{t-1}, that are computed for each of the 14 features. The previous N states are also stacked together to form a history of the previous observations, attempting to adhere to the Markov property. The final state served to the agent will then have $N \times |Features| \times |Stats|$ features. This totals up to 980 different features.

N is considered a hyperparameter of the problem; table II has a complete list of all hyperparameters used for MARLIN. For the choice of N, we followed the empirical considerations made in [24], and chose 10 as the length of the history after some preliminary trials and evaluations. During the training process, observations are processed and normalized through a moving average by using the *VecNormalize* environment wrapper present in SB3.

2) Actions: MARLIN takes continuous actions contained in the range $[-1, 1]$, which represent the percentage gain of the CWND size. For example, if the action chosen by the agent is going to be 1, the CWND is doubled; a value of 0 means no change in the CWND; -0.5 reduces the window size by 50%. The initial CWND size is set to 4KB at the beginning of the episode, as per the transport protocol implementation default.

For the purpose of this study, we capped the CWND size to 50 KB, a value that, if reached, would yield double the throughput that the network can accommodate in the setup we used for training and evaluation. This choice only impacts the very first phases of training, when the agent takes random steps to prime the model. After this phase, which in MARLIN is set to last 10K steps, our experiments have shown that the agent correctly never fills the CWND to 50 KB.

3) Reward: We designed a reward function that gives higher rewards to the agent the closer it gets to fully utilizing the available bandwidth:

$$r_t = -\frac{target_t}{target_t + acked\_kilobytes_t^{cumulative}} \tag{1}$$

where $target_t$ represents the amount of bytes the agent should have delivered up to step t since the beginning of the episode in order to fully utilize the link and $acked\_kilobytes_t^{cumulative}$ represents the number of kilobytes there were acknowledged by the receiver until step t.

A strictly negative reward function promotes the agent to accumulate the smallest amount of penalties. The penalty received is much smaller the closer the agent it is, at each step, to having utilized the link to the best of its possibilities.

The careful reader might notice that such rewarding system encourages the agent to accumulate acked bytes regardless explicit impact on the RTT, falling into the risk of privileging actions that could produce more acked bytes in the immediate future, with the drawback of causing undesired queuing delays. To prevent such risk, we consider a second formulation of the reward function that introduces a RTT-based penalty coefficient:

$$r_t = -\frac{target_t}{target_t + acked\_kilobytes_t^{cumulative} * (1 - penalties)} \tag{2}$$