MVFST-RL: An Asynchronous RL Framework for Congestion Control with Delayed Actions

Viswanath Sivakumar[1], **Tim Rocktäschel**[1,2], **Alexander H. Miller**[1], **Heinrich Küttler**[1], **Nantas Nardelli**[1,3], **Mike Rabbat**[1], **Joelle Pineau**[1,4], and **Sebastian Riedel**[1,2]

[1]Facebook AI Research
[2]University College London
[3]University of Oxford
[4]McGill University & MILA

Abstract

Effective network congestion control strategies are key to keeping the Internet (or any large computer network) operational. Network congestion control has been dominated by hand-crafted heuristics for decades. Recently, Reinforcement Learning (RL) has emerged as an alternative to automatically optimize such control strategies. Research so far has primarily considered RL interfaces which *block* the sender while an agent considers its next action. This is largely an artifact of building on top of frameworks designed for RL in games (*e.g.* OpenAI Gym). However, this does not translate to real-world networking environments, where a network sender waiting on a policy without sending data is costly for throughput. We instead propose to formulate congestion control with an asynchronous RL agent that handles delayed actions. We present MVFST-RL, a scalable framework for congestion control in the QUIC transport protocol that leverages state-of-the-art in asynchronous RL training with off-policy correction. We analyze modeling improvements to mitigate the deviation from Markovian dynamics, and evaluate our method on emulated networks from the Pantheon benchmark platform. The source code is publicly available at https://github.com/facebookresearch/mvfst-rl.

1 Introduction

Congestion control is one of the key components of any large computer network, most notably the Internet, and is crucial to enable operation at scale.[1] The goal of a congestion control algorithm is to dynamically regulate the rate of data being sent at each node to maximize total throughput and minimize queuing delay and packet loss. The vast majority of network strategies still rely on hand-crafted heuristics that are reactive rather than predictive. An early method, called Remy [14], demonstrated that offline-learned congestion control can be competitive with conventional methods. More recently, RL-based approaches have been proposed and show promise in simulated environments [6, 13].

Despite the above advances, to our knowledge, no RL method has been transferred to real-world production systems. One of the biggest drawbacks of RL congestion control from a deployment perspective is that policy lookup takes orders of magnitude longer compared to hand-crafted methods. Moreover, networking servers often have limited resources to run a machine learning model, thus requiring inference to be offloaded to dedicated servers and further delaying action updates. Current RL congestion control environments follow the synchronous RL paradigm (e.g. using the OpenAI

---

[1]It is estimated that 150,000 PB of data were sent per month over the Internet in 2018 (`https://www.statista.com/statistics/499431/global-ip-data-traffic-forecast/`).

Gym [2] interface), where model execution blocks the environment (network sender). This makes it infeasible for deployment where a sender waiting on a synchronous RL agent for congestion control, even for a few milliseconds, negatively impacts throughput (Figure 1).

In this paper we introduce MVFST-RL, a training framework that addresses these issues with a non-blocking RL agent for congestion control. For training in the presence of asynchronous interaction between the environment and the learner (owing to the inability of the environment to wait for gradient updates), we leverage IMPALA [3] which uses importance-sampling for off-policy correction. MVFST-RL is built on MVFST[2], a C++ implementation of the QUIC transport protocol used in Facebook production networks, allowing seamless transfer to deployment. To emulate real-world network traffic in RL environments for training, we build upon Pantheon [15] which obviates the need for hand-written traffic patterns and network topologies. We evaluate training with delayed actions and present our results on the Pantheon test-bed.
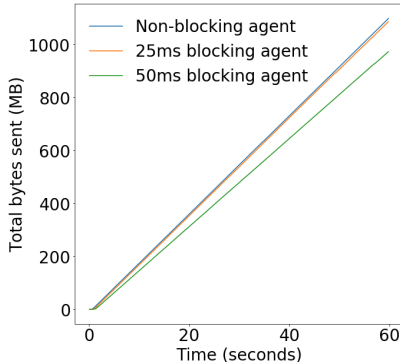

Figure 1: Cumulative bytes sent during 60 seconds over AWS California to Mexico emulator. RL agents which block the sender for 25ms and 50ms waiting on the policy transmit 1.1% and 11.4% fewer bytes, respectively, compared to the non-blocking agent. All agents take actions at 100ms intervals.

2 Related Work

A few different studies have applied RL to congestion control in varying ways. Iroko [13] takes the approach of a centralized policy to regulate sending rates of all the nodes in a network topology. While this is applicable for small to medium sized networks, it is intractable for large or Internet-scale networks. Iroko also requires manually specifying network topologies and traffic generators, making it difficult to emulate real-world network conditions. PCC-RL [6] implements congestion control as an environment in OpenAI Gym [2], with a blocking RL agent. Park's [8] congestion control environment based on CCP [10] is closest to MVFST-RL in design with Remote Procedure Calls (RPC) for environment–agent communication, but it effectively takes a synchronous approach with its short step-time of 10ms, constraining to only very small models.

3 Congestion Control as MDP with Delayed Actions

Consider a Markov Decision Process (MDP) [12] formulated as $\langle S, A, T, R \rangle$ where S is a set of states, A is a set of actions, R is the reward obtained, and $T(s'|s, a)$ is the state-transition probability function. Unrolling over time, we have the trajectory $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, ...)$. Casting congestion control as an MDP, the state space includes network statistics gathered from acknowledgement packets (ACKs) from the receiver such as round-trip time (RTT), queuing delay, packets sent, acknowledged and lost. The action is an update to the congestion window size, *cwnd*, which is the number of unacknowledged packets that are allowed to be in flight. Large *cwnd* leads to high throughput, but also increased delay due to queuing at intermediate buffers and network congestion. Our choice of action space A is a discrete set of updates to *cwnd*: $\{cwnd, cwnd/2, cwnd-10, cwnd+10, cwnd \times 2\}$ (Appendix A.2). The reward is generally a function of measured throughput and delay, aiming to strike a trade-off.

When an agent acts asynchronously on the environment at fixed time-intervals, the action a_t is applied after a delay of δ such that $t < t + \delta < t + 1$, where δ is the policy lookup time. The environment meanwhile would have transmitted further data based on the old action a_{t-1} during the interval $[t, t + \delta]$. The next state s_{t+1} therefore depends not just on s_t and a_t, but also on a_{t-1}. To incorporate this into the MDP, the state space can be augmented with the last action taken. In addition to this, given that our action space is relative to the previous actions taken, we find that a longer history of actions in the environment state helps learning. Therefore, we augment our state space with an action history of generic length k, which we treat as a hyperparameter, and obtain our augmented state

---

[2]https://github.com/facebookincubator/mvfst

space $\hat{S} = S \times A^k$. At time t, the augmented state $\hat{s}_t$ is $(s_t, a_{t-1}, ..., a_{t-k})$. With this formalism, we define our environment state and reward function below.

Environment State: Our choice of state vector contains a summary of network statistics gathered over a 100ms window along with a history of actions taken. From each ACK within the window, we gather 20 features such as the latest and minimum RTT, queuing delay, bytes sent, bytes acknowledged, and bytes lost and re-transmitted. The complete set of features are listed in Appendix A.1. To normalize the features, the time fields are measured in milliseconds and scaled by 10^{-3} and byte fields are in KB. We follow Copa's [1] strategy of computing *RTTStanding* to reduce noise in delay measurements. For each feature, we calculate sum, mean, standard deviation, min and max within a window, and concatenate them with k one-hot vectors of past actions along with the *cwnd* on applying those actions. This results in a state vector of size $100 + k \times (|A| + 1)$, where A is the action space defined earlier.

Reward: Our reward is simply $t - \beta \, d$ where t and d are the average throughput in MB/sec and the maximum delay in milliseconds, respectively, during the 100ms window, and the parameter β trades-off between the two. We experimented with logarithmic scales, such as Copa's [1] optimization function of $\log t - \beta \log d$, but we found it difficult to attain higher throughput such as 100 Mbps on faster networks when trained jointly with bad network scenarios such as cellular connections with 0.5 Mbps throughput.

In the following sections, we describe a framework for asynchronous RL training of congestion control with delayed actions and provide experimental analysis of the MDP with the augmented state space.

4   Introducing MVFST-RL

Framework: MVFST-RL follows an environment-driven design with an asynchronous RL agent. It integrates three components: (1) MVFST, Facebook's implementation of the QUIC transport protocol used in production servers, (2) TorchBeast[3][7], a PyTorch [11] implementation of IMPALA [3] for fast, asynchronous, parallel RL training, and (3) Pantheon [15] network emulators calibrated with Bayesian Optimization [9] to match real-world network conditions. The RL congestion controller accumulates network statistics from ACKs over a fixed time-window, say 100ms, and sends a state update asynchronously to an RL agent in a separate thread. Once the agent executes the policy, the action to update *cwnd* is applied asynchronously to the transport layer (Figure 2a).

IMPALA was originally designed to overcome scaling limitations during training of batched Actor-Critic methods, where parallel actors act for T steps each to gather trajectories (sequences of s_t, a_t, r_t, s_{t+1}), and then wait for a gradient update on the policy by the learner. IMPALA relaxes this restriction, and with an off-policy correction algorithm called V-trace, it allows actors to continue with trajectories while the learner asynchronously updates the policy. Although the original motivation was higher training throughput, the decoupled actor-learner algorithm and off-policy correction is well-suited for MVFST-RL, allowing the actors corresponding to network senders to not block on gradient updates, as doing so would change the dynamics of the underlying network.

Figure 2b illustrates the training architecture. Each actor corresponds to a separate Pantheon instance in a sender–receiver setup with a randomly chosen emulated network scenario. The senders run the QUIC protocol with RL congestion controller and communicate state updates via RPC to the actors. The actors execute the model for the next action, and once a full trajectory is obtained, they communicate it to the learner for a gradient update. All communications are asynchronous and do not block the network sender. After training, the model can be exported with TorchScript and executed in C++ on the network host without the need for RPC.

Model: Our model trunk is a two-layer fully-connected neural network with 512 layers each and ReLU non-linearity. The extracted features and the reward are fed into a single-layer LSTM [5]. For correctness during training, along with a trajectory of T steps, an actor also communicates the LSTM hidden state at the beginning of the trajectory to the learner. Finally, the LSTM output is fed into two heads – a policy head with $|A|$ outputs, and another head for the policy gradient baseline.

---

[3]https://github.com/facebookresearch/torchbeast

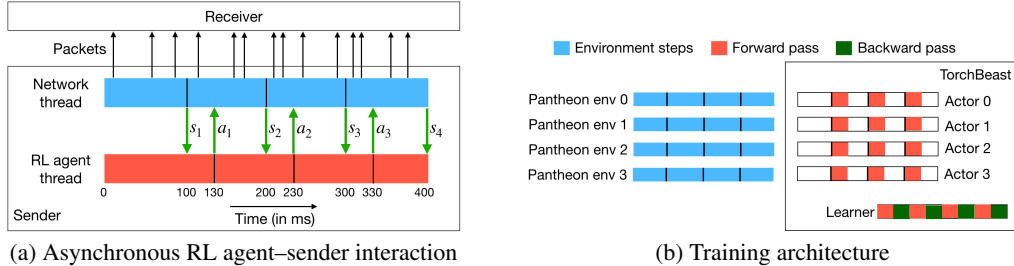(a) Asynchronous RL agent–sender interaction   (b) Training architecture

Figure 2: (a) States are sent to the agent every 100ms and policy lookup takes 30ms. In the synchronous paradigm, packets could not be sent during the 30ms sections between s_t and a_t. (b) TorchBeast maintains one actor per Pantheon environment, with RPC for state and action updates. The key difference from IMPALA is that, in addition to the actor–learner interaction, the environment–actor interaction is also asynchronous and the environment steps are not blocked by the forward-pass.

5   Experiments

Training: Our training setup consists of 40 parallel actors, each corresponding to a randomly chosen calibrated network emulator. We train episodically with each Pantheon sender–receiver instance running for 30 seconds, with the episodic trajectory used as a rollout for training. Episodic training provides the opportunity to learn dynamics similar to TCP Slow-Start, where the behavior of the algorithm during startup is sufficiently different from steady state. Following hyperparameter sweeps, we set the initial learning rate to 10^{-5} with RMSProp [4] and the trade-off parameter β in reward to 0.2. All experiments are run for a total of 10^6 steps.

Results: Figure 3a shows the test performance on a Pantheon emulator calibrated against Nepal to AWS India. MVFST-RL achieves competitive performance with higher throughput and similar delays compared to traditional methods like Cubic, Copa and NewReno, as well as better performance over learned methods like TaoVA. Compared to `rl-random`, a baseline agent that picks a random action over the same action space A at each step, our model consistently performs better. We also test against a network emulator unseen during training where it outperforms most other methods (Figure 3b).

One challenge we faced is generalization when training jointly over networks with widely different ranges of throughput and delay, causing rewards to be scaled very differently among the actors. We experimented with reward-clipping strategies as well as reward functions with logarithmic components as in Copa [1], but they result in policies that struggle to achieve high throughput when trained jointly with low throughput networks. This suggests that further research into reward shaping and multi-task learning strategies might be warranted.

Ablations: We examine the impact of the length of the action history in the augmented state space. Figure 3c plots the total reward obtained per episode, over the course of training, for varying values of the action history length k. While models with $k = 0$ or 1 struggle to learn, optimal performance is attained with $k = 10$ with diminishing returns beyond that. We also run the same experiment without LSTM and find the performance to be significantly worse, especially for larger values of k (Figures 3d and 3e). The results show that the MDP with the augmented state space performs better with delayed actions, and validate the need for a recurrent policy in such a setting.

6   Conclusion and Future Work

RL is a promising direction for improving real-world systems. Yet, it is challenging to deploy RL in datacenters due to performance and resource constraints, especially when evaluated against hand-written heuristics. We take the first steps towards moving away from blocking RL agents for systems problems and introduce a framework for learning asynchronous RL policies in the face of delayed actions. When applied to congestion control, our initial results show promise in the MDP with augmented state space. One challenge we face is learning a joint policy for a range of network scenarios when reward scales vary wildly due to differences inherent in the environment. We believe this is an interesting direction for further research on reward normalization strategies across environments. We hope to evaluate our trained congestion control agents in production networks in the future by lever-
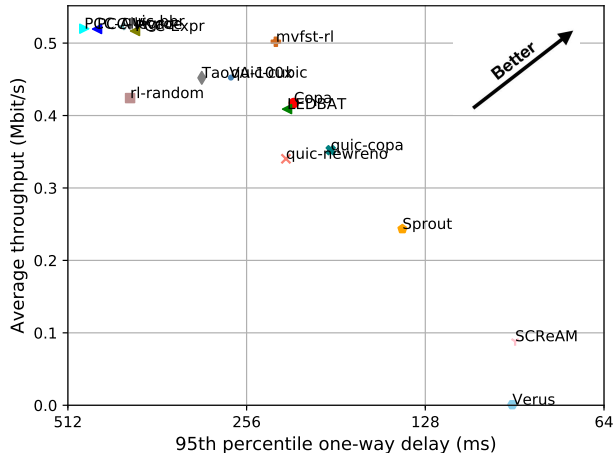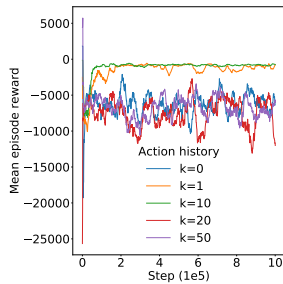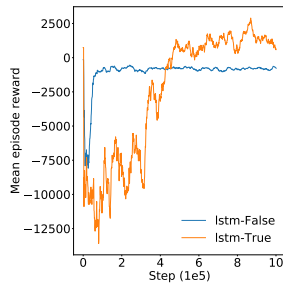
(a) Nepal to AWS India (calibrated)
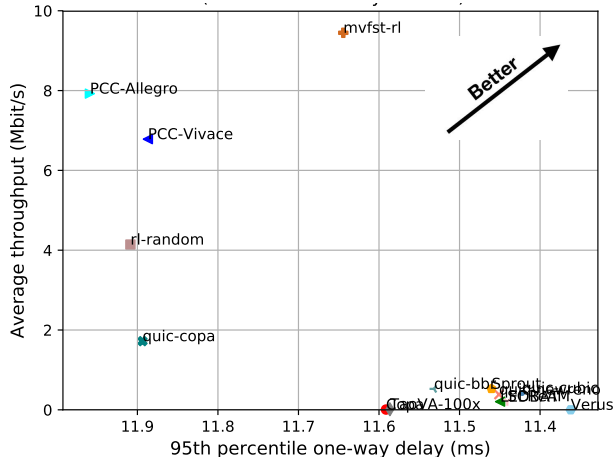
(b) Token-bucket policer (unseen environment)
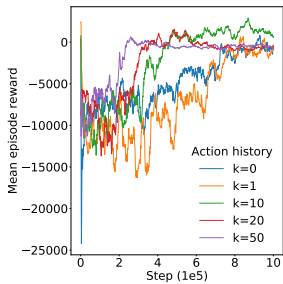
(c) With LSTM

(d) Without LSTM

(e) Action history = 10

Figure 3: Experimental Results. (a) and (b): Average throughput and 95th percentile delay when tested against calibrated Pantheon emulators. `rl-random` refers to a baseline agent that picks a random action over the same action space as `mvfst-rl` at each step. Mean of three 30-second runs. (c) – (e): Training rewards with and without LSTM for varying values of action history length k.

5

aging MVFST-RL's efficient implementation and tight integration with the QUIC transport protocol. MVFST-RL has been open-sourced and is available at https://github.com/facebookresearch/mvfst-rl.

Acknowledgments

References

[1] V. Arun and H. Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, Renton, WA, Apr. 2018. USENIX Association.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[3] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[4] G. Hinton, N. Srivastava, and K. Swersky. Overview of mini-batch gradient descent, 2012.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[6] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. A deep reinforcement learning perspective on internet congestion control. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3050–3059, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[7] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette. Torch-Beast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*, 2019.

[8] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani, S. He, V. Nathan, F. Cangialosi, S. B. Venkatakrishnan, W.-H. Weng, S. Han, T. Kraska, and M. Alizadeh. Park: An open platform for learning augmented computer systems. In *ICML Reinforcement Learning for Real Life Workshop*, 2019.

[9] J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pages 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[10] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. Restructuring endpoint congestion control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 30–43, New York, NY, USA, 2018. ACM.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[13] F. Ruffy, M. Przystupa, and I. Beschastnikh. Iroko: A framework to prototype reinforcement learning for data center traffic control. *CoRR*, abs/1812.09975, 2018.

[14] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 123–134, New York, NY, USA, 2013. ACM.

[15] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, July 2018. USENIX Association.

A Appendix

A.1 State Space

We design our state space to contain aggregate network statistics obtained from ACKs within each 100ms window (the environment step-time) and an action history of length k. An optimal value of k is chosen by hyperparameter sweeps to be 20. When an ACK or a packet loss notification arrives, we gather the following 20 network statistics:

	Feature	Description
1	lrtt	Latest round-trip time (RTT) in milliseconds (ms)
2	rtt_min	Minimum RTT (in ms) over a 10 second window
3	srtt	Smoothed RTT (in ms)
4	rtt_standing	Min RTT (in ms) over window of size $srtt/2$ [1]
5	rtt_var	Variance in RTT (in ms) as observed by the network protocol
6	delay	Queuing delay measured as $rtt\_standing - rtt\_min$
7	cwnd_bytes	Congestion window in bytes calculated as $cwnd * MSS$
8	inflight_bytes	Number of bytes sent but unacknowledged (cannot exceed $cwnd\_bytes$)
9	writable_bytes	Number of writable bytes measured as $cwnd\_bytes - inflight\_bytes$
10	sent_bytes	Number of bytes sent since last ACK
11	received_bytes	Number of bytes received since last ACK
12	rtx_bytes	Number of bytes re-transmitted since last ACK
13	acked_bytes	Number of bytes acknowledged in this ACK
14	lost_bytes	Number of bytes lost in this loss notification
15	throughput	Instantaneous throughput measured as $cwnd\_bytes/rtt\_standing$
16	rtx_count	Number of packets re-transmitted since last ACK
17	timeout_based_rtx_count	Number of re-transmissions due to Probe Timeout (PTO) since last ACK
18	pto_count	Number of times packet loss timer fired before receiving an ACK
19	total_pto_count	Total number of times packet loss timer fired since last ACK
20	persistent_congestion	Flag indicating whether persistent congestion is detected by the protocol

As a way of normalization, the time-based features (features 1–6) are scaled by 10^{-3} and byte-based ones (features 7–14) are scaled by 10^{-4}. Since there could be varying number of ACKs within any 100ms time window corresponding to an environment step, we obtain a fixed-size state vector by computing aggregate statistics from all ACKs within a window. This is achieved by computing sum, mean, standard deviation, min and max over each of the 20 features and flattening the resulting 100 features into a single vector. For features 1–9, it doesn't make sense to compute sums and we set those entries to zero in the state vector.

The history at time u, $u < t$, is encoded as a one-hot vector of the action a_u along with the congestion window as a result of that action $cwnd_u$. To the aggregate features, we concatenate a history of length k, resulting in our final state vector of length $100 + k \times (|A| + 1)$, where $|A|$ is the size of the action space. In our experiments, $k = 20$ and $|A| = 5$, resulting in a state vector of length 220.

A.2 Action Space

While there are various possible choices for the action space A, we chose the simplest set of discrete updates common in conventional congestion control methods:

$$A = \{cwnd, cwnd/2, cwnd - 10, cwnd + 10, cwnd \times 2\}$$

where $cwnd$ is the congestion window in units of Maximum Segment Size (MSS). The network environment starts with an initial $cwnd$ of 10, and bounded updates are applied according to the policy as follows:

$$cwnd_{t+1} = \text{clip}(\text{update}(cwnd_t, a_t, A), 2, 2000)$$

where a_t is the action according to the policy at time t and is an index into the action space A, $\text{update}(cwnd_t, a_t, A)$ is a function that updates the current congestion window according to a_t, and the function $\text{clip}(x, low, high) = \min(\max(x, low), high)$ bounds the congestion window to reasonable limits.

MVFST-RL supports configuring the action space to any discrete set of updates easily via a simple format. The above mentioned action space can be configured as `"0,/2,-10,+10,*2"`.