Finite Observation Histories versus Recurrent Memory in PPO-Based Congestion Control: A Controlled Simulation Study

Mubarak Abubakar Bello
Department of Computer Science and Engineering, Vivekananda Global University, Jaipur, Rajasthan, India

24wtec3csml003@vgu.ac.in

Abstract
Internet congestion control is a partially observable control problem: senders infer network conditions from delayed and noisy feedback. Reinforcement-learning controllers therefore require a temporal state representation, but the practical performance influence of learned recurrent memory relative to finite observation histories remains unclear. This study presents a controlled simulation-based comparison of three feed-forward Proximal Policy Optimization (PPO) policies using observation histories of 3, 5, and 10 monitor intervals and a recurrent PPO policy with long short-term memory (LSTM). The policies share the same network simulator, observation features, reward formulation, training protocol, and five-seed evaluation schedule. We evaluate them under flat-bandwidth, step-bandwidth, and cross-round-trip-time (cross-RTT) conditions using throughput, latency, packet loss, send-rate variability, reward, and inference latency. The resulting findings indicate an architectural trade-off among the compared policies, rather than a universal best policy.

Keywords: Internet congestion control; reinforcement learning; Proximal Policy Optimization; temporal state representation; observation stacking; recurrent neural networks; LSTM

1. Introduction

Internet congestion control is a sequential decision problem under partial observability. A sender does not directly observe bottleneck capacity, queue occupancy, or the behaviour of competing traffic. Instead, it acts on delayed and noisy signals derived from acknowledgements, including delivery rate, delay, and loss. The sender must therefore use information from more than one observation to decide whether to increase or decrease its sending rate. [1, 2]

Reinforcement learning (RL) offers a natural framework for this setting because it can learn a rate-control policy from interaction with a network environment. Learning-based controllers such as Aurora demonstrated that policy-gradient methods can optimize a throughput-delay objective from monitor-interval observations. Other

© The Author(s) 2026
A. Shastri et al. (eds.), *Proceedings of the Fourth Workshop on Applications of Computational Intelligence and Interactive Systems (WACIIS 2026)*, Advances in Intelligent Systems Research 217,
https://doi.org/10.2991/978-94-6239-832-0_20

works have continuously expanded RL-based congestion control to different objectives, deployment contexts, and network settings. However, the representation of temporal information is often treated as an implicit design detail or chosen alongside several other design decisions. As a result, it remains difficult to determine whether an observed benefit comes from an RL algorithm, reward formulation, training regime, or temporal-state representation. [3–7]

Two common approaches encode temporal information in different ways. Observation stacking concatenates the most recent measurements into a fixed-size state for a feed-forward policy. It is transparent and inexpensive, but it assumes that a bounded window contains the relevant context. A recurrent policy instead updates a hidden state over time and can, in principle, retain information over a variable horizon. Recurrent memory is attractive for partially observable problems, yet it also introduces sequential computation and a more difficult credit-assignment problem during training. Whether that additional complexity is worthwhile for congestion control is an empirical question. [6, 12]

This paper studies that question through a controlled simulation comparison. We evaluate PPO policies using fixed histories of 3, 5, and 10 monitor intervals against a recurrent PPO policy using LSTM memory. The comparison fixes the simulator, observation features, reward, training budget, and five-seed schedule as far as the two PPO implementations permit. We evaluate the policies under nominal flat-bandwidth conditions, a mid-episode bandwidth reduction, and an RTT shift. The objective is not to claim that one architecture universally dominates; rather, it is to characterize the throughput, congestion, stability, and inference-cost trade-offs induced by temporal-state design in this setting.

The paper makes four contributions:

1. It provides a controlled temporal-representation ablation for PPO-based congestion control, comparing three finite observation histories with recurrent LSTM memory.
2. It reports five-seed evaluation results across nominal, capacity-shift, and RTT-shift scenarios, separating throughput from congestion-related and stability-related outcomes.
3. It treats independently trained policies as the primary replication unit and reports seed-level uncertainty, while retaining episode-level rollout summaries as secondary evidence.
4. It measures the inference-time distribution of each architecture and identifies a consistent host-specific overhead associated with the recurrent policy.

The results show that the answer is scenario-dependent. In the evaluated configuration, finite histories, especially 5 and 10 monitor intervals, can provide favourable loss, stability, reward, and inference-cost trade-offs, while LSTM retains a throughput advantage in some conditions. The remainder of the paper reviews related work, describes the controlled design, presents the results, and discusses the limits of their generality.

2. Related Work

2.1 Learning-based congestion control

Classical congestion-control algorithms, including CUBIC and BBR, use hand-designed control rules based on congestion signals and bandwidth-delay estimates. Learning-based approaches instead adapt a control policy from observed outcomes. PCC and PCC Vivace framed congestion control as online performance optimization, while Pensieve established deep RL for adaptive video control. Aurora subsequently formulated congestion control as a deep RL problem and used PPO to learn monitor-interval rate adjustments. Later systems have considered real-world learning, hybrid designs, multi-objective objectives, and specialized deployment contexts. [1–11]

This literature establishes RL as a viable congestion-control paradigm, but the present work does not attempt to outperform every conventional or learning-based controller. Its narrower contribution is an architectural ablation within a shared PPO simulation framework. Classical-controller comparisons are an important next step for assessing absolute control quality, but they are not required to answer the representation question posed here.

2.2 Temporal state under partial observability

Temporal state is necessary when a current observation cannot fully identify the operating condition. Fixed-history policies make this state explicit by concatenating recent observations. This mechanism appears in many RL-based control frameworks, including congestion-control systems that use monitor-interval histories. Recurrent networks is an alternative method where an LSTM maps a sequence of observations into a hidden state that can retain relevant information beyond a manually selected window. [6, 10, 12, 13]

Earlier RL studies show that none of the approaches dominates across all partially observable problems. The usefulness of memory depends on the observation process, temporal horizon, optimization regime, and task dynamics. Some architecture benchmark frameworks have compared feed-forward, recurrent, transformer, and state-space models under common training protocols. Congestion control has different characteristics of delayed-feedback and queueing dynamics, so the suitability of recurrent memory should be measured in this domain rather than assumed from generic RL benchmarks.

2.3 Robustness and operational cost

Generalization is particularly important for learned congestion-control policies because training environments cannot cover every bandwidth, RTT, queue, and cross-traffic condition. Domain randomization and benchmarking frameworks address this concern, but the influence of temporal-state architecture on robustness is less often isolated. The cross-RTT and bandwidth-step experiments in this paper are deliberately limited tests of that influence; they are not a claim of broad out-of-distribution generalization. [14, 15]

Operational cost is also relevant because congestion control makes decisions continuously. Feed-forward stacked histories increase input dimension, whereas recurrent policies add a hidden-state update and sequential dependency. Existing RL congestion-control work has noted deployment overhead, but direct comparisons of temporal-representation cost remain uncommon. We therefore report timing as a supporting systems measure, with host-specific interpretation.

2.4 Position of this study

The contribution of this study is a bounded empirical result: in one controlled PPO congestion-control simulator, finite histories sometimes match or improve the reward, congestion, and stability trade-offs of recurrent memory at lower measured inference cost. The study complements, rather than replaces, work on new congestion-control algorithms, broad benchmark suites, and real-network deployment. Benchmarking platforms and protocols such as Pantheon and CC-Bench are complementary routes to broader validation [16, 17].

3. Experimental Design

3.1 Study design

We conducted a controlled simulation study of temporal state representation in reinforcement-learning-based congestion control. The comparison comprised three feed-forward Proximal Policy Optimization (PPO) policies that receive explicit histories of recent network observations and one recurrent PPO policy that maintains long short-term memory (LSTM). The study was designed to test whether recurrent memory provides benefits beyond a finite observation history when the simulator, observations, reward, training budget, and seed schedule are held as consistent as practicable.

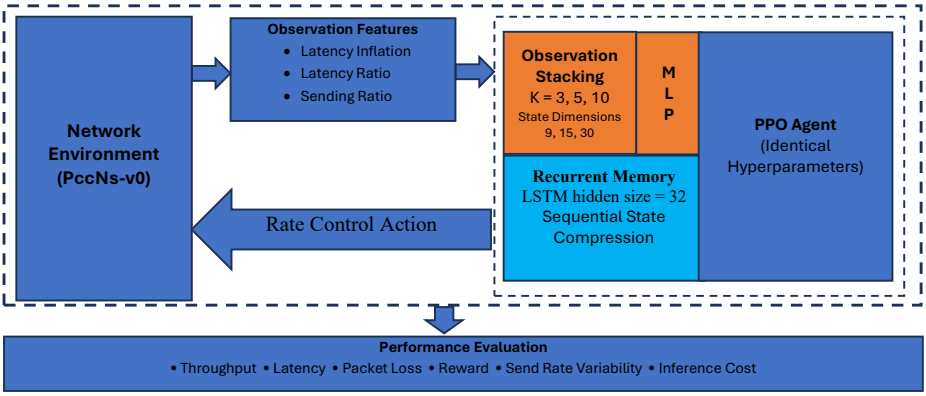

Figure 1. Experimental framework used to isolate the effect of temporal state representation on PPO-based congestion control. All models share identical PPO hyperparameters, reward functions, training budgets, and network environments, differing only in the mechanism used to encode temporal context.

The experimental factor was the representation used to incorporate temporal information. The feed-forward policies used histories of $k \in \{3, 5, 10\}$ monitor-interval observations; the recurrent policy received one current observation and updated an LSTM hidden state. The comparison therefore isolates an architectural design decision, although it does not make the two implementations mathematically identical: the recurrent policy uses the RecurrentPPO implementation and has recurrent rollout and optimization dynamics that differ from the feed-forward PPO implementation.

3.2 Congestion-control environment

The experiments use a custom Gymnasium-compatible network simulator, 'PccNs-v0', derived from the Aurora-style monitor-interval control formulation. A single sender transmits over a simulated path and adjusts its sending rate once per monitor interval. The baseline training configuration uses 200 Mbps bandwidth, 30 ms propagation latency, zero background loss, a queue capacity of five bandwidth-delay products, and 1500-byte packets. [19]

At decision time (t), the sender observes three normalized signals derived from acknowledgement feedback: latency inflation, latency ratio, and sending ratio. A single observation is therefore three-dimensional. For stacked policies, the state is the concatenation of the most recent (k) observations, yielding input dimensions of 9, 15, and 30 for $k = 3$, $k = 5$, and $k = 10$, respectively. The LSTM policy instead processes the current three-dimensional observation sequentially and carries a learned hidden state.

The action is a continuous rate-adjustment factor, where (a_t) is the scalar continuous action output by the policy, (0.025) is the configured action scale, (x_t) is the sender's current rate, and the simulator clips the sending rate to (40,1000)

$$\boldsymbol{\delta_t = 0.025 \cdot a_t,\ a_t \in x_t}$$

$$\boldsymbol{x_{t+1} = \begin{cases} clip_{[40,1000]}\left(x_t(1+\delta_t)\right), & \delta_t \geq 0 \\ clip_{[40,1000]}\left(\frac{x_t}{(1-\delta_t)}\right), & \delta_t < 0 \end{cases}}$$

All policies use the same reward implemented in the simulator, which rewards received throughput and penalizes latency and loss:

$$\boldsymbol{r_t = 0.001\left(\left[\frac{10 \cdot throughput}{8 \cdot packet\ size}\right] - 1000 \cdot Latency - 2000 \cdot Loss\right)}$$

The exact units and normalization follow the simulator implementation. The reward should be interpreted as the study's common optimization objective rather than as an independent network-quality measure.

3.3 Policy configurations and training

The feed-forward policies use Stable-Baselines3 PPO with MLP policies. The recurrent policy uses 'sb3-contrib' RecurrentPPO with an 'MlpLstmPolicy', LSTM hidden size 32, and single 32-unit policy and value layers. The stored model-configuration table reports 9,731, 10,499, and 12,419 trainable parameters for the k = 3, k = 5, and k = 10 stacked policies, respectively, and 11,651 for the LSTM policy. [7, 20]

Each model was trained for 5,000,000 timesteps for each of five seeds: 2, 7, 13, 18, and 24. The checked-in training configuration uses four vectorized simulation environments, 400 rollout steps, batch size 400, learning rate (3×10^{-4}), discount factor 0.99, GAE parameter 0.95, clipping range 0.2, value-function coefficient 0.5, entropy coefficient 0, and maximum gradient norm 0.5. The reported experiments were conducted in a Kaggle Linux environment with four logical Intel Xeon CPU cores, 31.35 GiB of memory, and two Tesla T4 GPUs (15 GB each). The captured software environment used Python 3.12.12, PyTorch 2.10.0+cu128, CUDA 12.8, cuDNN 9.10.2, Stable-Baselines3 2.8.0, sb3-contrib 2.8.0, Gymnasium 0.29.1, and NumPy 1.26.4. The active training entry point selects CUDA when available; CUDA was available in the recorded environment. Environment simulation and vectorized environment management remain CPU-side operations.

3.4 Evaluation protocol

Each trained model was evaluated for ten episodes under each scenario, producing 50 evaluated episodes per model-scenario cell across the five trained seeds. The scenarios were:

Table 1. Evaluation scenarios.

Scenario	Bandwidth	Propagation latency	Purpose
Flat	200 Mbps	30 ms	Same nominal parameters as training
Step	200 to 100 Mbps within the episode	30 ms	Capacity-change response
Cross-RTT	200 Mbps	80 ms	Response to an RTT shift

For every episode, we computed mean throughput, mean latency, mean packet-loss rate, mean reward, and the within-episode standard deviation of sending rate as a send-rate-instability proxy. The 50 episodes are fresh simulator rollouts, but ten rollouts share each trained policy. The primary experimental unit was therefore the independently trained seed, not an individual episode. We first averaged episode metrics within each seed and scenario and then compared the resulting five seed-level values for each policy. Episode-level summaries are reported as secondary within-policy variability information.

We report seed-level means, standard deviations, and paired bootstrap 95% confidence intervals for differences between each stacking policy and LSTM. The bootstrap used 20,000 resamples of the five paired seed differences. This small number of training replications warrants cautious interpretation; confidence intervals are used to communicate uncertainty rather than to support broad claims from large numbers of correlated episode observations.

3.5 Inference-time measurement

Inference cost was measured by recording the elapsed wall-clock time around each deterministic ‘model.predict()’ call during evaluation. The resulting data contain 20,000 calls for each model-scenario group (five seeds, ten episodes per seed, and 400 decision steps per episode). We report the mean, median, and tail percentiles, together with per-seed summaries. Because calls within an episode are serial measurements on the same host, they are not treated as 20,000 independent deployment trials.

The timing run was performed in the GPU-enabled Kaggle environment described above. The active implementation was configured to select CUDA when available, and CUDA was available in the recorded environment. However, the archived timing record does not preserve the loaded policy's device assignment at the moment of each ‘model.predict()’ call. No explicit warm-up exclusion or CUDA synchronization was applied in the recorded measurement. The results should therefore be interpreted as comparative measurements from this host and implementation, rather than as GPU-kernel latency measurements or universal deployment-latency estimates.

4. Results

4.1 Overview

All results in this section are regenerated from the 60 raw evaluation logs in 'logs/eval/'. Each value aggregates ten evaluation episodes within each of five independently trained seeds. The results emphasize effect direction and seed-level uncertainty rather than episode-level p-values. Figures are used to visualize training and trajectory behaviour; they do not change the replication unit or supply additional independent observations.

Across the three scenarios, recurrent memory did not produce a uniform advantage. LSTM often produced the highest throughput, whereas stacking k=5 and k=10 frequently operated at lower latency and loss, with lower send-rate variability and higher reward. The pattern was strongest in the flat and cross-RTT scenarios and more mixed after the bandwidth step.

4.2 Training dynamics

Figure 2 provides descriptive training-monitoring evidence across the five training seeds. The three stacking variants rise rapidly and settle near an episode-reward plateau of roughly 480–500 EWMA units, whereas the recurrent policy settles at a lower mean level and exhibits a visibly wider late-training band. The k=5 and k=10 curves reach their plateau earlier than k=3 in the plotted monitor. This is not an independent inferential test: training workers, updates, and adjacent monitoring points are correlated. Its role is to show that the final evaluation trade-offs were not obtained from a visibly non converged stacking policy.

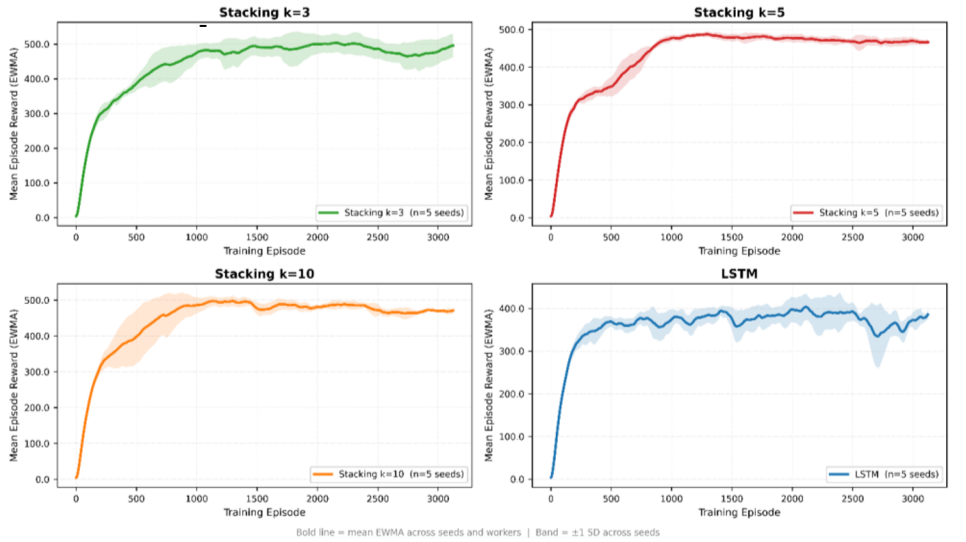

Figure 2. Training reward convergence by temporal representation. Bold traces are mean EWMA reward across the five seeds; shaded bands show the dispersion plotted by the source figure. This is descriptive training-monitoring evidence.

4.3 Aggregate operating points

Table 2 reports mean values across the five seeds. Throughput is shown alongside congestion and stability metrics because the principal result is a trade-off, not a single winning model.

Table 2. Mean metrics across five trained seeds.

Scenario	Model	Throughput (Mbps)	Latency (s)	Loss rate	Send-rate instability	Reward
Flat	LSTM	1.547	0.0703	0.081	0.268	1.056
Flat	Stacking k = 3	1.729	0.0697	0.152	0.654	1.067
Flat	Stacking k = 5	1.741	0.0706	0.116	0.519	1.148
Flat	Stacking k = 10	1.613	0.0678	0.075	0.203	1.126
Step	LSTM	1.149	0.0682	0.098	0.634	0.276
Step	Stacking k = 3	1.374	0.0719	0.259	0.612	0.555
Step	Stacking k = 5	1.256	0.0696	0.259	1.124	0.460
Step	Stacking k = 10	1.289	0.0700	0.227	0.315	0.550
Cross-RTT	LSTM	2.012	0.1720	0.140	0.393	1.224
Cross-RTT	Stacking k = 3	2.095	0.1738	0.196	0.695	1.179
Cross-RTT	Stacking k = 5	1.921	0.1692	0.095	0.319	1.242
Cross-RTT	Stacking k = 10	1.780	0.1689	0.086	0.224	1.144

4.4 Flat-bandwidth evaluation

In the flat scenario, k=10 achieved lower mean latency, lower loss, lower send-rate instability, and higher reward than LSTM, while LSTM had higher mean throughput. The paired seed-level difference for k=10 minus LSTM was -0.0065 s for latency (bootstrap 95% CI [-0.0117, 0.0015]), -0.279 for loss rate ([-0.381, -0.122]), -0.990 for send-rate instability ([-1.710, -0.314]), and +0.392 for reward ([+0.215, +0.614]). Its throughput difference was -0.207 Mbps ([-0.513, +0.091]), so the data do not support a clear throughput difference at this replication level.

The k=3 and k=5 policies also had higher mean reward than LSTM, but their seed-level uncertainty intervals for loss and latency included zero. These results make k=10, rather than all stacked policies indiscriminately, the strongest flat-condition result.

4.5 Cross-RTT evaluation

The cross-RTT scenario produced the clearest separation between recurrent and finite-history policies. Compared with LSTM, k=5 had lower latency (-0.0063 s, [-0.0111, -0.0015]), lower loss (-0.285, [-0.472, -0.070]), lower send-rate instability (-1.046, [-2.046, -0.069]), and higher reward (+0.407, [+0.176, +0.626]). The k=10 policy showed the same direction for latency, loss, instability, and reward. Its throughput was lower by 0.345 Mbps ([-0.724, -0.053]), whereas the k=5 throughput interval included zero.

LSTM achieved the highest mean throughput in this scenario. The results therefore indicate a control trade-off: in the tested RTT shift, finite histories, particularly k=5 and k=10, reduced congestion-related penalties and improved the common reward objective, while recurrent memory could pursue a more aggressive throughput operating point.

Figure 2 gives the corresponding time-resolved loss trajectories. In the cross-RTT panel, the LSTM trace rises more strongly over the episode than the stacking traces; the latter remain lower, with k=10 visually among the lowest trajectories. This visual pattern is consistent with the seed-level loss comparison, but it is descriptive because monitor intervals within an episode are serially dependent.

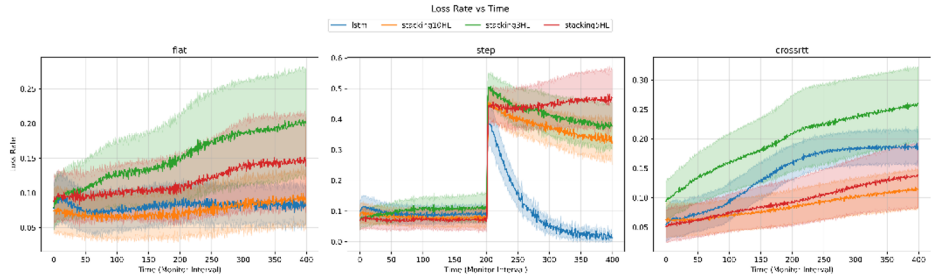

Figure 3. Loss-rate trajectories by evaluation scenario and temporal representation. The figure illustrates within-episode dynamics; it is interpreted alongside seed-level aggregate results, not as independent monitor-interval evidence.

4.6 Step-bandwidth evaluation

After the bandwidth step, no temporal representation dominated every metric. The k=3 policy achieved the highest mean throughput (1.374 Mbps), whereas k=10 had the lowest mean loss (0.227) and lowest send-rate instability (0.315). Relative to LSTM, k=10 reduced send-rate instability by 0.565 ([-0.977, -0.242]) and increased reward by 0.275 ([+0.145, +0.400]). Its latency, loss, and throughput intervals included zero at the seed level, with no clear differences. The k=5 policy had a higher mean send-rate instability than LSTM in this scenario.

The step-change result here should be interpreted as evidence that the k=10 history can provide a more stable adaptation in this configuration, and should not be understood as evidence that finite observation histories are uniformly more robust to bandwidth changes.

The send-rate trajectory (Figure 4) helps interpret this result. Under flat and cross-RTT conditions, LSTM follows a higher send-rate trajectory than the stacked policies, especially later in the episode. This aggressive operating point justifies its higher throughput and higher loss. In the step panel, all curves respond at the intervention point, but the stacked traces, particularly k=10, remain below the recurrent trace afterwards. These trajectories illustrate a control-policy difference, the instability statistic and seed-level intervals, rather than the plotted lines alone, support the stability claim.

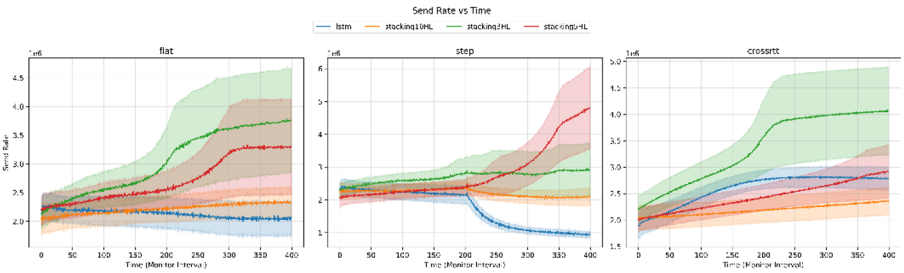

Figure 4. Send-rate trajectories by evaluation scenario and temporal representation. The curves visualize the operating-point dynamics described in the text.

Figure 5 integrates the throughput and loss dimensions at the episode level. Its cloud of points makes the non-dominance visible: LSTM occupies a higher-throughput region in several conditions but also extends to higher-loss regions, while k=10 clusters toward lower loss at a cost in throughput. The plot visualizes the operating-point trade-off; it does not estimate an independent sample size from its individual points.

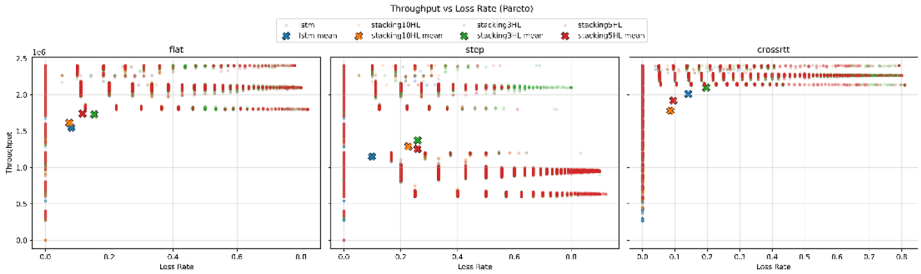

Figure 5. Episode-level throughput versus loss-rate operating points. Markers identify plotted model means. Individual episode points are shown for visualization and are not treated as independent training replications.

4.7 Inference cost

The timing distributions showed a consistent descriptive difference between feed-forward and recurrent policies. Mean inference time for the three stacking policies was 0.618–0.629 ms in the flat and cross-RTT scenarios and 0.626–0.632 ms after the bandwidth step. LSTM required 1.028 ms, 1.021 ms, and 1.014 ms, respectively. Thus, within the benchmark environment, the stacked policies used approximately 38–40% less mean decision time than LSTM.

Increasing the explicit history length from 3 to 10 observations did not materially increase measured decision time. This finding is consistent with the small network sizes in the comparison, and should be treated as host- and implementation-specific until repeated under a fully documented timing protocol.

Figure 6 shows the full empirical timing distributions. Across all three scenarios, the stacked-policy curves lie to the left of the LSTM curve through most of the distribution, including the plotted p95 and p99 reference levels. Thus, the mean-time reduction is not solely a consequence of a few extreme calls. These calls are repeated

measurements on the same evaluation host; the figure is descriptive of this implementation environment rather than a device-independent latency guarantee.

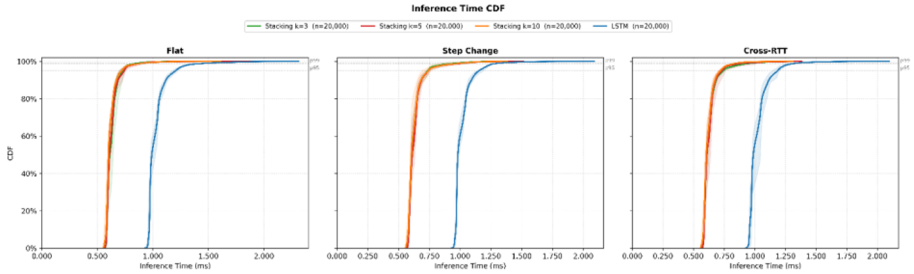

Figure 6. Empirical cumulative distributions of per-call inference time by model and scenario. The plotted p95 and p99 lines are host-specific descriptive timing references.

5. Discussion

5.1 Temporal representation impacts the policy operating point

The resulting contrast shows that there is no universally best temporal representation. Instead, it shows that architecture changes the operating point selected by the learned controller. LSTM achieved higher throughput in most cases, particularly in the cross-RTT scenario, but this is observed alongside higher packet loss and send-rate variability. The k = 5 and k = 10 stacked policies accepted some throughput reduction in exchange for lower congestion-related penalties and higher reward under the common training objective. This distinction matters because throughput alone is not sufficient to characterize congestion-control quality.

The strongest result is not that every finite history is superior in every scenario. In flat scenario, k = 10 showed lower latency and loss, lower send-rate instability, and higher reward than LSTM at the seed level. Under an RTT shift, both k = 5 and k = 10 showed this pattern, although k = 10 had lower throughput. After the bandwidth step, k = 10 maintain a stability and reward advantage, while the other comparisons were mixed. These differences indicate that history length is itself a design choice where adding some explicit context can help, but no single horizon could be assumed optimal without testing.

5.2 A bounded history may already provide sufficient context

The simulator exposes only three monitor-interval features, and the control interval adapts to observed latency. In this setting, a short history may already capture the trend information needed to respond to queue growth and delivery changes. An LSTM has greater representational flexibility, but flexibility does not guarantee a better learned control policy. Its hidden state must be learned from finite trajectories with

delayed feedback, and the recurrent implementation changes the optimization dynamics as well as the representation.

This explanation is an interpretation, not a direct causal measurement of the hidden state. The study does not show that LSTM memory is intrinsically harmful, nor does it identify the temporal horizon required by a real network. It shows that, for this observation design and PPO configuration, recurrent complexity did not provide a consistent benefit large enough to dominate the measured trade-offs.

5.3 Inference cost and deployment relevance

The stacking policies show a consistent advantage over LSTM variant in the measured decision-time distributions. Their mean decision times were approximately 38–40% lower than LSTM, and increasing the history from 3 to 10 observations had little effect on the measured mean. This supports a practical design principle where a finite-history feed-forward baseline should be evaluated before adopting recurrent memory solely to address partial observability.

The timing result is not a deployment guarantee. It depends on the host, software stack, timer, model implementation, and batching behaviour. A production congestion-control implementation would also need to consider integration overhead, concurrency, memory pressure, operating-system scheduling, and network-stack interaction. The present result establishes a comparative overhead within the experimental environment, not an end-to-end deployment latency bound.

5.4 Limitations

Several limitations bound the conclusions. First, the experiments use a single-flow simulator and a restricted set of bandwidth and RTT conditions; they do not address multi-flow fairness, cross traffic, or real-network implementation. Second, only PPO/RecurrentPPO and one LSTM configuration were studied. The feed-forward and recurrent implementations share central hyperparameters but cannot be made identical in every optimization detail. Third, the study uses five independently trained seeds. The ten episodes evaluated per seed improve measurement precision for each policy but do not replace additional independent training runs. Fourth, the reward and three-feature observation design may favour particular behaviours and may not transfer to a different control objective. Finally, the inference timing was measured on a specific Kaggle T4-based environment and requires complete run-environment reporting for reproducibility.

5.5 Future work

Future work should test the finding with more training seeds, independently controlled evaluation trajectories, multi-flow and cross-traffic settings, and real-network or emulation environments. It should compare finite histories with alternative memory mechanisms, such as gated recurrence, attention, or state-space models, under matched capacity and training budgets. Finally, a preregistered evaluation

protocol with seed-aware hierarchical analysis would make cross-architecture claims more robust.

6. Conclusion

This study examined finite observation histories and learned recurrent memory as temporal-state representations in the PPO-based Internet congestion control framework. Across five trained seeds and three simulated evaluation scenarios, recurrent LSTM memory did not provide a uniform advantage. Finite histories, particularly $k = 5$ and $k = 10$, often achieved lower loss, lower latency or send-rate variability, higher reward, and lower measured inference cost. LSTM retained higher throughput in some conditions, showing that the outcome is a trade-off rather than a universal ranking.

Within the evaluated simulator and observation design, finite observation stacking is therefore a strong temporal baseline for PPO congestion control. The result should be interpreted as an architecture and scenario-specific empirical finding. Broader conclusions require additional seeds, controlled evaluation randomness, and real-network validation.

Declarations

Conflict of interest

The author declares no conflict of interest.

Data and code availability

The source code, trained models, experimental configurations, raw evaluation logs, and analysis scripts used in this study are publicly available in the GitHub repository at: https://github.com/mbrk-tofa/Temporal_State_in_PPO_CC.git These materials provide the information necessary to reproduce the training, evaluation, and statistical analyses reported in this manuscript.

Acknowledgements
The author would like to express sincere appreciation to Dr. Manish Shrivastava, Head of the Department of Computer Science and Engineering, Vivekananda Global University Jaipur, for the valuable guidance, constructive feedback, and insightful discussions provided during the seminar stage of this research. His guidance helped shape the direction of the study, and his continued advice and encouragement throughout the research process contributed significantly to its successful completion.

Funding Statement

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] S. Ha, I. Rhee, and L. Xu, “Cubic,” *ACM SIGOPS Operating Systems Review*, 42(5), 64–74, 2008. doi:10.1145/1400097.1400105.

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-based congestion control,” *Queue*, 14(5), 20–53, 2016. doi:10.1145/3012426.3022184.

[3] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, “PCC: Re-architecting congestion control for consistent high performance,” in *NSDI*, 2015, pp. 395–408.

[4] M. Dong et al., “PCC Vivace: Online-learning congestion control,” in *NSDI*, 2018, pp. 343–356.

[5] H. Mao, R. Netravali, and M. Alizadeh, “Neural adaptive video streaming with Pensieve,” in *SIGCOMM*, 2017, pp. 197–210. doi:10.1145/3098822.3098843.

[6] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, “A deep reinforcement learning perspective on Internet congestion control,” in *ICML*, 2019, pp. 3050–3059.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv:1707.06347, 2017.

[8] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, “QTCP: Adaptive congestion control with reinforcement learning,” *IEEE Transactions on Network Science and Engineering*, 6(3), 445–458, 2019. doi:10.1109/TNSE.2018.2835758.

[9] L. Zhang, Y. Cui, M. Wang, K. Zhu, Y. Zhu, and Y. Jiang, “DeepCC: Bridging the gap between congestion control and applications via multiobjective optimization,” *IEEE/ACM Transactions on Networking*, 30(5), 2274–2288, 2022. doi:10.1109/TNET.2022.3167713.

[10] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, “Owl: Congestion control with partially invisible networks via reinforcement learning,” in *INFOCOM*, 2021, pp. 1–10. doi:10.1109/INFOCOM42981.2021.9488851.

[11] R. Galliera, A. Morelli, R. Fronteddu, and N. Suri, “MARLIN: Soft actor-critic based reinforcement learning for congestion control in real networks,” in *NOMS*, 2023, pp. 1–10. doi:10.1109/NOMS56928.2023.10154210.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, 9(8), 1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

[13] H. Tian, X. Liao, C. Zeng, J. Zhang, and K. Chen, “Spine: An efficient DRL-based congestion control with ultra-low overhead,” in *CoNEXT*, 2022, pp. 261–275. doi:10.1145/3555050.3569125.

[14] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, “A study on overfitting in deep reinforcement learning,” arXiv:1804.06893, 2018.

[15] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, “Quantifying generalization in reinforcement learning,” in *ICML*, 2019, pp. 1282–1289.

[16] F. Y. Yan et al., “Pantheon: The training ground for Internet congestion-control research,” in *USENIX ATC*, 2018, pp. 731–743.

[17] S. Abbasloo, “Internet congestion control benchmarking,” arXiv:2307.10054, 2023.

[18] K. Winstein and H. Balakrishnan, “TCP ex machina,” in *SIGCOMM*, 2013, pp. 123–134. doi:10.1145/2486001.2486020.

[19] M. Towers et al., “Gymnasium: A standard interface for reinforcement learning environments,” *Advances in Neural Information Processing Systems*, 38, 2025.
[20] A. Raffin et al., “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, 22(1), 12348–12355, 2021.
[21] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in *NeurIPS*, 2019.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

