Adaptive Network Traffic Shaping with Deep Reinforcement Learning: A PPO-Driven Token Bucket on ns-3

Prince Kwabena Appiah Boadu
Department of Computer Engineering
Kwame Nkrumah University of Science and Technology
Kumasi, Ghana
pkappiahboadu@st.knust.edu.gh

***Abstract*—Static Token Bucket Filters (TBFs) enforce a single contracted rate regardless of how traffic actually behaves. Set the rate too low and the shaper starves bursts and triggers TCP retransmissions; set it too high and bursts pass through unsmoothed and congest downstream queues. Real internet demand is non-stationary, so any fixed rate is wrong for most of the day.**

We replace the fixed rate with a learned policy. A Proximal Policy Optimization (PPO) agent observes a four-dimensional vector (queue, throughput, drop intensity, demand) and emits a continuous TBF rate every second. The agent is trained inside a custom Gymnasium environment that drives an ns-3 bottleneck topology, using a four-level curriculum that progresses from constant CBR through bursty on-off, mixed elephant/mice flows, and finally three weeks of real Cloudflare Radar traffic. We evaluate two action shapes, a wide variant ($a \in [1, 100]$ Mbps) and a narrow variant ($a \in [40, 90]$ Mbps), against six static baselines spanning 30–80 Mbps.

On real-world demand, the wide-action agent holds queue occupancy and drops far below the aggressive 30 Mbps static cap; the narrow-action agent recovers most of the throughput of a high static cap at substantially lower rate oscillation and roughly 28% fewer drops than the 30 Mbps cap. Stress tests at 2× and 3× nominal demand show that both agents degrade gracefully, concentrating loss into narrow burst windows rather than spreading it uniformly across the trace.

The compact four-scalar observation is sufficient to learn a non-trivial rate-control policy without per-flow state, packet traces, or topology maps, which is significant because four scalars are exactly what a production shaper can realistically expose to a control plane. An RL-shaped TBF replaces a contract with the past (a fixed rate fitted to yesterday's profile) with a contract over states, generalising across diurnal cycles without operator intervention.

The action ceiling becomes the binding constraint under extreme overload, so dynamic action ranges and recurrent policies are the highest-impact next steps. Beyond the shaper itself, the most interesting direction is composition with adjacent ML controllers: pricing-driven demand shaping, application-layer encoders, and downstream router-queue controllers, jointly trained against a user-visible QoS reward.

***Index Terms*—Reinforcement learning, Traffic shaping, Token bucket filter, Network simulation, ns-3, PPO, Quality of service**

I. Introduction

Quality-of-Service (QoS) on a shared link is built on three primitives: traffic contracts, which fix a Committed Information Rate (CIR) and a burst allowance between a customer and a provider; policers, which monitor offered traffic and drop or re-mark anything in excess; and shapers, which absorb excess into a buffer and drain it at the contracted rate, smoothing bursts at the cost of latency [1], [2]. Among shapers, the Token Bucket Filter (TBF) remains the canonical implementation in Linux's `tc` and in most core-router QoS pipelines. Its parameters *rate* and *burst* are set once and assumed to be a faithful summary of the traffic profile they govern.

Real internet demand is not stationary. Cloudflare Radar shows backbone traffic oscillating between roughly 10% and 100% of the daily peak within a single 24-hour cycle [3] (Fig. 1). A single static rate is therefore wrong for most of the day. Set it to the mean and bursts overflow the bucket, triggering TCP retransmissions that further amplify congestion. Set it to the peak and most of the shaping action disappears, leaving downstream queues to absorb whatever arrives. The operator's "safe" middle is a moving target.

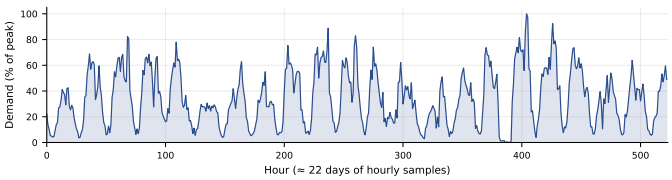

Fig. 1. Three weeks of normalized hourly Cloudflare Radar traffic for a representative ASN. Demand swings between roughly 10% and 100% of peak within a daily cycle. A static rate cap is correct only on the narrow band of hours where the curve crosses it.

This paper reports on an end-to-end system that replaces the fixed TBF rate with a continuously learned one. A PPO agent [4] observes a four-scalar summary of the bottleneck (queue, throughput, drop intensity, current demand), emits a continuous TBF rate in $[1, 100]$ Mbps once per second, and is trained inside a custom Gymnasium [5] environment that drives an ns-3 [6] bottleneck. The agent is trained with a curriculum that ends in real Cloudflare Radar traffic, then evaluated A/B against static rate caps under both nominal and 2–3× overload.

Our contributions are:

1) a clean, subprocess-isolated bridge between Stable-Baselines3 PPO [7] and ns-3 that supports parallel rollout workers;
2) a normalized, bounded multi-objective reward that keeps PPO's value head numerically well-conditioned across very different traffic regimes;
3) an A/B evaluation against six static baselines on real and stress-scaled traffic;

4) a precise account of where this approach holds and where it does not.

II. Background and Related Work

The shared theme across recent ML-for-networking work is that static, human-coded rules are no longer adequate for the dynamic regimes that modern traffic produces [8]. Reinforcement learning has become the dominant tool because the underlying control problems are sequential decisions under uncertainty, exactly the setting where trial-and-error in simulation pays off. We organise the most relevant prior work by the layer of the stack it touches.

Application layer. Pensieve [9] frames adaptive video bitrate selection as an MDP and learns a policy that beats hand-tuned heuristics. Ahmed et al. [10] push the same idea into 5G-VANET multimedia streaming, using distributed RL to jointly tune quantization, GoP size, and frame rate to match an unstable wireless channel. Thompson et al. [11] use real-time telemetry plus session metadata (player count, input frequency) to prioritise latency-critical game packets. All three optimise *what is sent* and how it is encoded.

Transport and scheduling layer. Jay et al. [12] treat congestion control itself as an RL problem at the endpoint. Liao et al. [13] target Time-Sensitive Networking, using DRL to compress Gate Control Lists by up to 69.5% so they fit in real switch memory while preserving deadlines. Wang et al. [14] hybridise Double Q-learning with particle swarm optimisation to escape local optima in NP-complete TSN-5G end-to-end scheduling.

Infrastructure and queueing layer. Iroko [15] is the closest in spirit to our work: an RL framework for prototyping data-centre traffic-control policies. Kattepur et al. [16] cast router port-queue configuration as a POMDP and let a model-based RL agent replace the manual "trial-and-error" tuning that 5G slicing makes intractable. Shames et al. [17], the earliest precedent we found, used a Q-learning agent with a small neural network to learn token-generation rates for a shaper from drop percentage and buffer occupancy. They showed that even a simple RL controller can outperform a fixed TBF when the topology shifts; our work updates this two-decade-old idea with modern policy optimisation, real Cloudflare demand, and ns-3 ground truth.

Demand-side and economic layer. SHIFT [18] takes a different lever altogether: a multi-agent DRL pricing scheme that shifts user demand spatially and temporally, flattening peak load without any new hardware. This complements rather than competes with shaping; pricing reduces the volume the shaper has to handle.

Where this paper fits. The papers above span the stack from application encoding to economic incentives, but they leave a gap at the egress-shaper layer: the single point where a contracted SLA is mechanically enforced. Shames et al. addressed this gap in 2006 with tabular Q-learning on a synthetic network; we revisit it with PPO, a real ns-3 simulator, and live Cloudflare demand traces. Compared to [16] and [15], our agent operates on a deliberately impoverished observation space (four scalars; no per-flow state; no topology map), which is what an operator can realistically expose from a production shaper without a deep telemetry pipeline. Compared to [17], we extend from a single static topology to a four-level curriculum that ends in real diurnal traffic, and we use clipped policy updates [4] to avoid the cascade of TCP timeout storms that an unconstrained update (e.g. rate jumping from 90 to 10 Mbps in one step) would otherwise trigger.

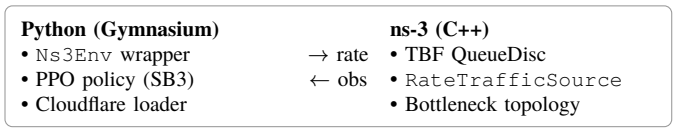

Fig. 2. Subprocess bridge between the Python policy and the C++ simulator. Each control interval is a fresh ns-3 process invoked with the agent's chosen rate.

What this paper does. Concretely, we (i) build a reproducible PPO + ns-3 + Gymnasium loop with a one-shot subprocess bridge that removes the long-lived-simulator failure modes of prior bespoke integrations; (ii) propose a normalized, bounded multi-objective reward that keeps the value head well-conditioned across very different traffic regimes; (iii) characterise the failure modes of static TBFs across the 30–80 Mbps range under real Cloudflare demand; and (iv) provide an A/B comparison of two action shapes (wide and narrow) under nominal and 2–3× overload, including an honest account of where the learned policy degrades.

III. System Design

A. Architecture

The system is a tight feedback loop between a Python policy and a C++ simulator (Fig. 2). At each control interval, the Gymnasium environment emits an action (a TBF rate), spawns ns-3 as a one-shot subprocess with that rate as a CLI argument, ingests one CSV line of telemetry from the subprocess's stdout, and returns the next observation and reward.

```
./ns3-sim --rate=52.5Mbps --burst=6500000
--source=80.0Mbps --duration=1.0
```

This subprocess model is deliberate. Every step is a fully isolated process with no shared state, no port-binding conflicts, and no long-lived simulator state to corrupt. Parallel rollout workers come for free: the only contention is the OS process table.

B. ns-3 Bottleneck

The simulated topology is a single bottleneck link of 100 Mbps physical capacity, with the TBF queue discipline acting as the logical bottleneck where shaping happens. Senders inject traffic via ns-3's `OnOffHelper` at a CBR rate that is itself driven from the Cloudflare hourly demand series, so the agent's input traffic mirrors a real ASN profile. Telemetry is sampled exclusively at the egress of the TBF (queue depth, throughput, and drop count), so the control policy sees only what an operator could plausibly export from a real shaper.

TABLE I

TRAINING CURRICULUM. EACH LEVEL ADDS ONE NEW FAILURE MODE THAT THE PREVIOUS LEVEL DID NOT EXPOSE.

Level		Traffic	Demand	Focus
1)	Basic	Constant CBR	60 Mbps	Rate–demand match
2)	Bursty	On–Off	60 Mbps	Temporal awareness
3)	Chaotic	Elephant + mice	80 Mbps	Noise robustness
4)	Real	Cloudflare Radar	Real-world	Generalization

The TBF *rate* is rewritten every second from the agent's action. The *burst* is derived as a fixed multiple ($0.1 \cdot \text{rate} \cdot 1$ s) of the chosen rate, which keeps the bucket window at a sensible 100 ms of the controlled rate and avoids adding a second action dimension that the agent would otherwise need to learn jointly.

C. Observation, Action, Reward

The observation is normalized to $[0, 1]^4$:

$$o_t = \left(\frac{q_t}{Q_{\max}}, \frac{T_t}{T_{\max}}, \tanh\left(\frac{d_t}{d_{\max}}\right), \frac{D_t}{D_{\max}} \right), \tag{1}$$

where q_t is queue occupancy (bytes), T_t throughput (Mbps), d_t drop count over the interval, and D_t the current demand. Bounds ($Q_{\max} = 5$ MB, $T_{\max} = 100$ Mbps, $d_{\max} = 100$, $D_{\max} = 100$ Mbps) are physical link properties or simulator caps and are kept constant across curriculum levels. The agent never sees raw byte counts or unscaled rates; it sees pressure signals.

The action space is a single continuous variable, the TBF rate in Mbps. We compare two action shapes:

- **Wide**: $a_t \in [1, 100]$ Mbps (the full physical range);
- **Narrow**: $a_t \in [40, 90]$ Mbps (the operationally realistic band).

The reward combines three terms:

$$R_t = \frac{\alpha T_{\text{norm}} - \beta Q_{\text{norm}} - \gamma D_{\text{norm}}}{\alpha + \beta + \gamma} \tag{2}$$

with $\alpha = 1.0$, $\beta = 0.5$, $\gamma = 0.8$. Throughput and queue are min–max scaled against their bounds; drops are passed through $\tanh$ rather than scaled linearly so the gradient remains sharp near zero (where it must distinguish "rare drops" from "no drops") and saturates at large values (so a buffer overflow during early training does not produce a -100 reward that derails the value function). Dividing by $\alpha + \beta + \gamma$ keeps $R_t \in [-1, +1]$, which decouples PPO's learning rate from the choice of weights.

D. Curriculum Training

The agent is trained across four progressively harder environments (Table I), with weights carried forward from one level to the next. Each level adds one new failure mode that the previous level did not expose.

For hyperparameter search and rapid iteration we additionally use a mock simulator based on an Ornstein–Uhlenbeck process [19],

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t, \tag{3}$$

TABLE II

PPO AND ENVIRONMENT HYPERPARAMETERS. VALUES SHARED ACROSS CURRICULUM LEVELS EXCEPT WHERE NOTED.

Parameter	Value
Discount γ	0.99
PPO clip range	0.2 (0.15 at level 4)
Learning rate	3e-4 (5e-5 at level 4)
Rollout length n	512 (256 at level 4)
Minibatch size	64
GAE λ	0.95
Entropy coefficient	0.0
Reward weights (α, β, γ)	(1.0, 0.5, 0.8)
Burst multiplier	0.1 of rate
Control interval	1 s
Episode length	50 steps
Queue ceiling $Q_{\max}$	5 MB
Drop normalizer $d_{\max}$	100
Action range (Wide)	$[1, 100]$ Mbps
Action range (Narrow)	$[40, 90]$ Mbps

Input: policy π_θ, value head V_φ, curriculum $\mathcal{C} = (C_1, C_2, C_3, C_4)$, ns-3 binary E
for level C in $\mathcal{C}$ **do**
  for $i = 1, ..., n$ **do**
    sample action $a_t \sim \pi_\theta(\cdot \mid o_t)$
    spawn EC with rate a_t, demand D_t, $\Delta t = 1$ s
    read $(q_{t+1}, T_{t+1}, d_{t+1})$ from stdout
    compute R_t via (2); store (o_t, a_t, R_t, o_{t+1})
  update (θ, φ) via PPO clipped objective
transfer (θ, φ) to next level

Algorithm 1. Curriculum PPO training loop. Each environment step is a fresh ns-3 subprocess; rollout buffers are populated in parallel across multiple workers.

with $\mu = 70$ Mbps, $\theta = 0.10$, $\sigma = 20$ Mbps, plus a 20% per-step probability of an additive 20–50 Mbps spike. The mock is roughly two orders of magnitude faster than ns-3 and produces trajectories that drift, spike, and recover in patterns that genuinely resist memorization.

Hyperparameters used for the runs reported in this paper are collected in Table II. Levels 1–3 train for 50–80 k environment steps each; level 4 (real Cloudflare traffic) trains for an additional 50 k steps starting from the level-3 weights. Total wall-clock training time on a 16-core commodity workstation is approximately 9 hours when running 8 ns-3 rollouts in parallel.

IV. FAILURE MODES OF STATIC SHAPING

Before introducing the agent, we map how a static TBF behaves under real Cloudflare demand (Fig. 3). The story is a clean trichotomy.

Low-rate starvation (30 / 40 Mbps). Throughput is capped well below available demand for most of the cycle. The bucket overflows in peak hours (Fig. 3 b), and cumulative drops climb monotonically (Fig. 3 c). For TCP this triggers retransmissions

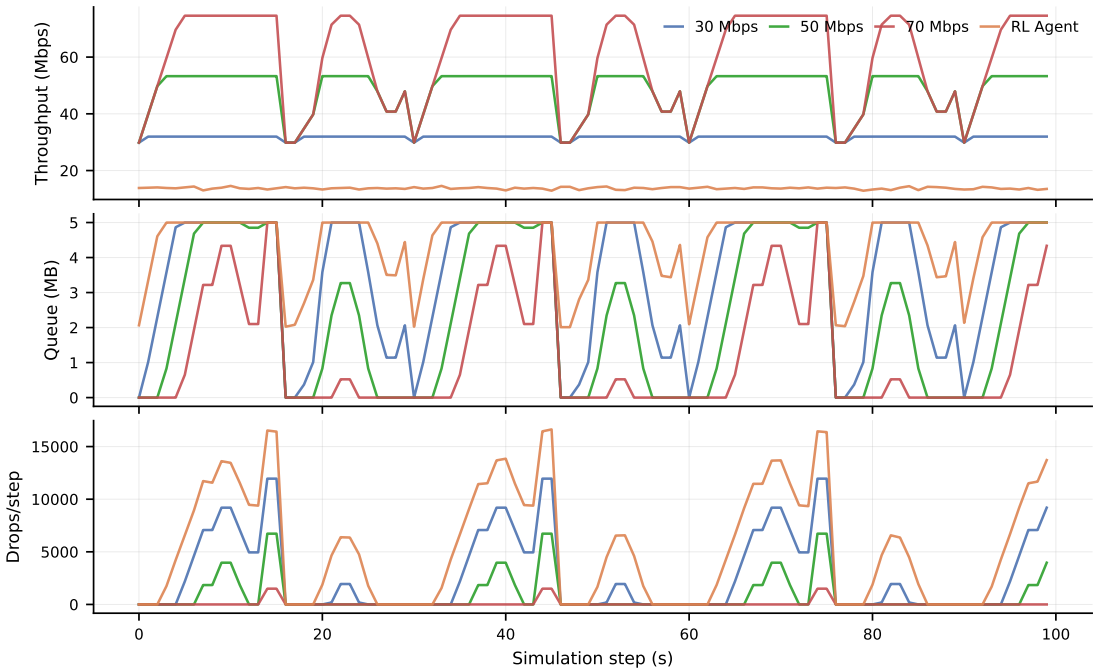

Fig. 4. Wide-action agent versus static baselines on the real-traffic window. Top: throughput; middle: queue occupancy; bottom: per-step drops. The 70 Mbps baseline tops the throughput ranking but at the cost of zero adaptive control.

that compound into a congestion spiral; for VoIP and video, the sustained queue delay is itself an SLA failure.

Middle-ground compromise (50 / 60 Mbps). The "operator default": eliminates the worst overflows but introduces a different failure mode. Off-peak hours leave significant idle capacity, and unexpectedly large surges still produce occasional drops. Drop counts are reduced an order of magnitude relative to 30 Mbps but are not zero.

High-rate permissiveness (70 / 80 Mbps). Drops essentially disappear, but the shaper has ceded any control over burst behaviour. During heavy-hitter periods the queue still builds, just more slowly, and the marginal throughput gain over 50 Mbps is smaller than the marginal control loss.

The narrow band of "acceptable" static rates therefore shifts with the time of day. This motivates an adaptive policy.

The trichotomy is also visible in classical operator playbooks: practitioners choose static rates by inspecting historic peak/95th percentile traffic and adding margin. That margin is wasted capacity when the link is idle, and insufficient margin when traffic exceeds the historic peak. The adaptive policy we propose makes this margin state-conditional rather than time-invariant.

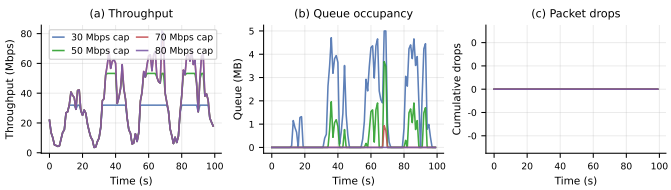

Fig. 3. Static TBF baselines on a 100-second window of real Cloudflare demand at four fixed rates. (a) Throughput is capped at the rate ceiling; (b) queue occupancy spikes at low rates as the bucket overflows; (c) cumulative drops climb sharply for the 30 and 40 Mbps caps and stay near zero only at 70 Mbps and above.

V. Evaluation

We evaluate two trained variants, Wide ($a \in [1, 100]$ Mbps) and Narrow ($a \in [40, 90]$ Mbps), against the 30, 50, and 70 Mbps static baselines. All runs use the same real-traffic input; the only difference is the rate-selection policy. Each window is 100 seconds of simulated time at one-second control intervals, repeated identically across scenarios so the demand trace is held constant.

A. Real Traffic, Nominal Intensity

Fig. 4 shows the wide-action agent's trajectory on the real-traffic window. Throughput tracks demand closely; the agent periodically dips slightly below current demand to flush queue buildup before it crosses into a drop event. Queue and drop traces are correspondingly bursty; the agent is active rather than conservative.

The narrow-action agent (omitted as a separate time-series figure for brevity) produces visibly smoother rate curves, fewer large corrections, and a tighter orbit around the demand signal, at the expense of slightly slower reaction to abrupt spikes.

Aggregate behaviour is shown in Fig. 5. Three takeaways:

1) The Wide agent's average throughput sits well below the 70 Mbps cap because it spends much of its time at rates slightly below current demand to keep queue depth low. It is the *strategy* that distinguishes the agent, not the headline throughput number.
2) The Narrow agent recovers most of the throughput while still cutting drops compared to the 30 Mbps static cap by roughly 28%.
3) Both agents trade some headline throughput for substantially reduced drop counts versus the low static

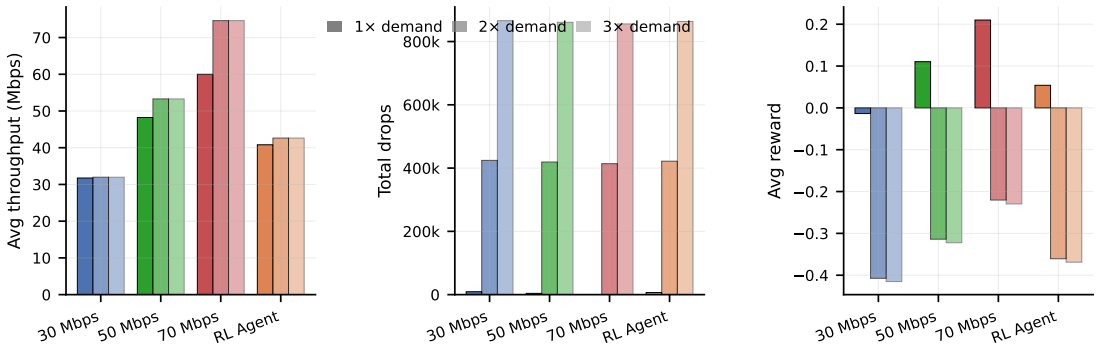

Fig. 6. Narrow-action agent under nominal (1×), 2×, and 3× demand. Throughput and reward both collapse for all policies as the link saturates, but drop counts rise uniformly across baselines, indicating loss in the saturated regime is dominated by physics rather than policy.

rates that an operator would choose if they were prioritising "safety" over capacity.

The crucial qualitative difference between the agent and the static baselines is not the average value on any single metric but the *shape* of the trajectories. The 70 Mbps cap maximises raw throughput by ignoring queue depth entirely; the agent's throughput is lower precisely because it actively flushes the buffer. This is a strategy choice, not a performance loss. An operator who values low end-to-end latency on a saturated link will read these numbers as a win for the agent; one who only contracts for raw bytes-per-second will not.

B. Stress: 2× and 3× Demand

The deployment question is generalisation. We scale the demand input by 2× and 3× beyond the training distribution (Fig. 6).

At 2× demand the link is approaching saturation. The agent's rate sits near the upper boundary of its action space far more frequently than under nominal traffic; momentary buffer builds appear in the queue trace but recover within a few control intervals without cascading into sustained loss. At 3× demand, physics wins: genuine link saturation produces drop counts on the order of 10^5 per window for every policy, including the agent. The agent does not eliminate loss in this regime; what it does is concentrate loss into narrow burst windows rather than spread it uniformly across the window. This is the honest limit of the design; a constrained action space cannot reach above its ceiling, and the narrow agent's ceiling becomes a real constraint.

VI. Discussion and Limitations

A. Engineering Notes from the Implementation

Two engineering decisions deserve highlighting because they materially affected reproducibility. First, our initial implementation kept ns-3 running as a long-lived subprocess and communicated rate updates over a UNIX pipe. This intermittently froze the simulator: ns-3 buffers its stdout in default mode, and the Python parent would block on a read that the child had not yet flushed. Switching to a one-shot subprocess per step eliminated the freeze entirely at a cost of about 30 ms of process-spawn overhead per step. Second, packet-counter inconsistencies between TCP and the TBF queue discipline (driven by ns-3′s segmentation behaviour at the sender) produced reward spikes during the first second of every episode. Resetting the counters at episode boundaries rather than relying on running totals solved the problem.

Roughly two-thirds of the development time on this project went into plumbing of this kind: stable telemetry, deterministic seeding across parallel workers, and CSV schema versioning to survive incremental changes to the ns-3 build. None of it is novel science, but it is where reproducibility lives.

B. What the Result Means

The compact four-scalar observation is the result we set out to validate, and the A/B evaluation supports it. The agent has no flow table, no DPI, no topology map. It distils an entire bottleneck into four numbers (queue, throughput, drop

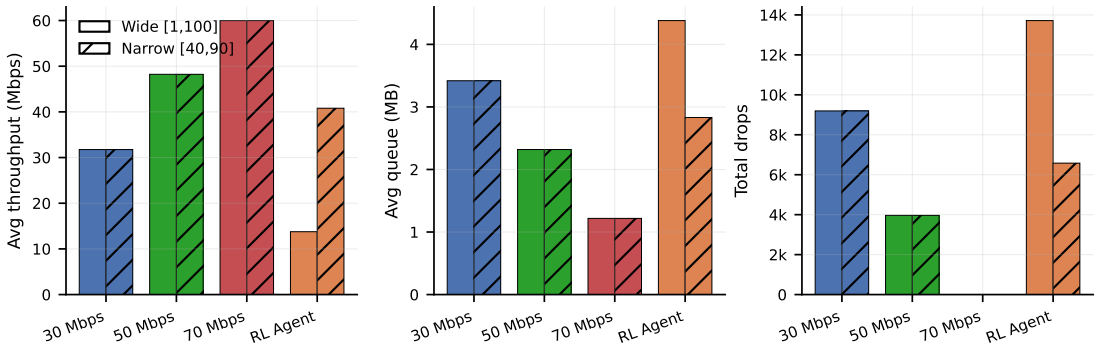

Fig. 5. Aggregate metrics across baselines and the two agents. Solid bars: wide action shape ($[1, 100]$ Mbps). Hatched bars: narrow action shape ($[40, 90]$ Mbps). Throughput, queue, and drops are reported as 100-second window averages or totals.

intensity, demand) and acts on them. That this is sufficient for non-trivial rate control on real traffic is the core finding.

The narrow-versus-wide comparison is more pragmatic than scientific: constraining the action space to the operationally realistic band trades exploration coverage for convergence speed and policy smoothness. In production, the narrow agent is the one we would deploy; the wide agent's wider exploration is principally a diagnostic for what the policy *would* do given the freedom.

Limitations. The single-bottleneck topology elides multi-tenant fairness; the egress-only telemetry assumes per-class metrics are exposed; and the per-step subprocess model adds a non-trivial fixed cost (≈ 30–80 ms wall-clock per step on commodity hardware) that limits how fast a deployed control loop can run. None of these are inherent to the approach; they are engineering choices that fit the simulator we had access to.

Future work. Three directions are worth pursuing in order of impact:

- **Recurrent policies.** An LSTM head over the four-scalar observation would give the agent an explicit short-term memory of recent queue trajectories, which the current MLP must reconstruct from instantaneous state.
- **Latency-aware reward.** Queue depth is a leading indicator of latency, but it is not latency. Adding a direct delay term to (2) would let the agent trade queue depth against actual end-to-end delay.
- **Dynamic action ceilings.** The 3× stress regime shows that a fixed upper bound on rate is the binding constraint at extreme overload. A policy that scales its action range with observed demand would handle this without retraining.
- **Composition with adjacent ML controllers.** The most interesting next step is composition rather than a bigger model. Pricing-based demand shaping [18] could feed our shaper a partially flattened arrival process; an application-layer RL encoder [10], [11] could co-adapt video or game bitrates against the rate our agent advertises; and a downstream router-queue controller [16] could absorb the residual bursts our shaper does not flatten. Each of these has been studied in isolation, but the joint training problem (multiple RL agents at different layers of the same path, sharing a reward that is anchored to user-visible QoS) is open.
- **Hardware-aware deployment.** Time-sensitive scheduling work like AGCS [13] and DQHPSO [14] shows that real switches impose hard memory and timing constraints that a software-only simulator hides. Porting our policy onto a programmable data-plane (P4 / DPDK) would expose those constraints and force a more honest cost model than the 30–80 ms subprocess overhead we currently pay.

VII. Conclusion

A static TBF is a contract with the past; it commits the network to a rate set under the assumption that yesterday's traffic profile is representative of today's. An RL-shaped TBF is a different kind of contract: it commits to a *policy* over states rather than a specific rate, and so handles non-stationary traffic without operator intervention. On real Cloudflare demand, our PPO agent delivers competitive throughput with substantially reduced drop counts versus the low and middle static rates that operators typically deploy. Under 2–3× overload it degrades gracefully rather than catastrophically. The four-scalar observation is enough, and the subprocess-isolated training loop is fast enough to make this a reproducible, single-machine experiment. Source code and data are available at https://github.com/blackprince001/network-shaping.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "RFC 2475: An Architecture for Differentiated Services," *IETF Request for Comments*, 1998.

[2] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Prentice Hall, 2010.

[3] Cloudflare, "Cloudflare Radar: Internet Traffic Insights." 2026.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] G. Brockman *et al.*, "OpenAI Gym." 2016.

[6] nsnam Project, "ns-3 Network Simulator." 2024.

[7] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[8] K. Dulaj and others, "Harnessing Machine Learning for Intelligent Networking in 5G Technology and Beyond: Advancements, Applications and Challenges," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 6, pp. 605–633, 2025, doi: 10.1109/OJITS.2025.3564361.

[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *Proceedings of ACM SIGCOMM*, 2017.

[10] A. A. Ahmed and others, "Smart Traffic Shaping Based on Distributed Reinforcement Learning for Multimedia Streaming over 5G-VANET Communication Technology," *Mathematics*, vol. 11, no. 700, pp. 1–20, 2023, doi: 10.3390/math11030700.

[11] M. R. Thompson and others, "AI-Enabled Traffic Shaping for Latency-Critical Game Sessions." 2024.

[12] N. Jay, N. H. Rotman, P. B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," in *International Conference on Machine Learning (ICML)*, 2019.

[13] J. Liao and others, "Adaptive Gating-Controlled Scheduling in Time-Sensitive Networks via Deep Reinforcement Learning," in *2nd International Conference on Electrical Automation and Artificial Intelligence (ICEAAI)*, 2026, pp. 948–952. doi: 10.1109/ICEAAI68945.2026.11442459.

[14] X. Wang and others, "Reinforcement Learning-Based Particle Swarm Optimization for End-to-End Traffic Scheduling in TSN-5G Networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 3254–3268, 2023, doi: 10.1109/TNET.2023.3276363.

[15] F. Ruffy, M. Przystupa, and I. Beschastnikh, "Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control," in *Workshop on ML for Systems at NeurIPS*, 2019.

[16] A. Kattepur and others, "Automated Configuration of Router Port Queues via Model-Based Reinforcement Learning," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6. doi: 10.1109/ICCWorkshops50388.2021.9473670.

[17] I. Shames and others, "Application of Reinforcement Learning in Development of a New Adaptive Intelligent Traffic Shaper," in *5th International Conference on Machine Learning and Applications (ICMLA)*, IEEE Computer Society, 2006, pp. 1–6.

[18] S. Choi and others, "SHIFT: Multi-Agent Reinforcement Learning for Spatiotemporal Mobile Traffic Shaping via Dynamic Pricing," *IEEE Transactions on Network and Service Management*, vol. 22, no. 6, pp. 6143–6158, 2025, doi: 10.1109/TNSM.2025.3618117.

[19] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Review*, vol. 36, no. 5, pp. 823–841, 1930.