Rediscovering heuristics: A litmus test for RL in networked systems

Seoyul Oh Yikai Zheng* Eric Xiang Gao Deepak Vasisht Francis Y. Yan

University of Illinois Urbana-Champaign **Tsinghua University*

ABSTRACT

Reinforcement learning (RL) holds promise for producing policies that outperform human-designed heuristics in networked systems, yet its training process remains brittle and opaque. We present a diagnostic methodology for RL training based on a key insight: strong heuristics in networked systems are typically governed by only a handful of parameters, so a compact RL policy class that is just expressive enough to represent a heuristic baseline should reliably *rediscover* it by learning to match its behavior and performance.

We argue that heuristic rediscovery serves as a litmus test for the RL training pipeline. While passing this test does not validate a deep neural policy, failing it exposes problems in training (e.g., a poorly chosen reward function or environment distribution) before additional model complexity obscures their causes. Moreover, the compact policy space makes the training process amenable to visualization and diagnosis. It enables us to inspect three views that are normally hidden: the return landscape, the optimization path, and the correctness properties of the learned policy.

Our evaluation showcases the value of heuristic rediscovery in adaptive bitrate streaming, revealing when and why RL training succeeds or fails. Experiments on congestion control further suggest the generality of our methodology.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; • **Networks** → **Network performance analysis**.

KEYWORDS

Reinforcement learning, RL training diagnosis, Heuristic rediscovery, Networked systems, Adaptive bitrate streaming, Congestion control

* Work done while visiting UIUC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:

Seoyul Oh, Yikai Zheng, Eric Xiang Gao, Deepak Vasisht, and Francis Y. Yan. 2026. Rediscovering heuristics: A litmus test for RL in networked systems. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

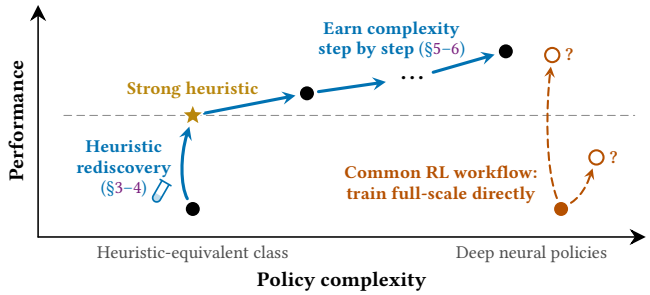


Figure 1: A common workflow commits to a full-scale policy class before training, learning only afterward whether the resulting policy outperforms a strong heuristic. Our methodology first trains a heuristic-equivalent policy as a litmus test; after rediscovering the heuristic, the pipeline incrementally introduces model complexity.

Reinforcement learning (RL) has emerged as a promising approach to developing control policies for networked systems. Unlike handcrafted heuristics, RL-based policies learn through experience and reward feedback, and have demonstrated strong potential in applications such as adaptive bitrate (ABR) streaming [16], real-time video [1, 29], resource scheduling [5, 17, 26], and congestion control [14, 21].

However, RL-trained policies often perform poorly after deployment [27] and exhibit counterintuitive behavior [6], hindering real-world adoption. To address this challenge, prior work has sought to interpret [12, 18, 22], verify [6, 28], or enhance [7, 25] a policy *after* initial training. In contrast, we focus on diagnosing the end-to-end training process itself to understand *why training arrived there.*

RL training is known to be "black magic" because its outcome depends on a complex interplay of design choices, such as the reward function, state representation, and environment distribution. Deep neural policies further complicate this by obscuring how these choices influence the learned behavior. As a result, when an RL policy underperforms, practitioners would simply sweep over training configurations with little insight into the underlying causes of failure. Richer diagnostic views are generally unavailable as the policy often resides in a high-dimensional parameter space, making training difficult to visualize and analyze.

Our experience with RL training motivates a new diagnostic methodology centered on the following question: *before trusting RL to discover a better policy, can it reliably rediscover a known, strong heuristic?* We argue that heuristic rediscovery serves as a practical litmus test for RL training (Fig. 1), providing both a minimum capability check and an instrument to visualize and diagnose the training pipeline.

The heuristic litmus test. Many time-tested controllers in networked systems are simple rule-based heuristics. Buffer-based ABR policies map the playback buffer level to a video bitrate using a few thresholds [11, 23]; loss- or delay-based congestion control algorithms respond to loss signals [9, 20] or RTTs [3, 4] according to predefined rules.

We observe that the core behavior of these heuristics can be captured by a compact RL policy class with only a handful of parameters. One such class is logistic or softmax regression, which *linearly* maps a heuristic's observed state (input features) to logits and then normalizes them into probabilities over actions via a sigmoid or softmax function. This insight suggests a minimum capability test for an RL training pipeline: if we restrict the policy space—including both the state-action space and the parameter space—such that it is just expressive enough to represent a heuristic, RL training should reliably produce a policy that matches the heuristic's behavior and performance in principle. We refer to this test as *rediscovering the heuristic* (§3.1).

Rediscovery does not mean recovering the heuristic's exact parameters or behavior cloning (no supervision from the heuristic is used). Rather, it means that a minimal but sufficiently expressive policy class, trained solely through reward-driven RL, can learn the heuristic's essential control structure and achieve comparable performance.

Passing this test establishes only a minimum bar: it does not guarantee that a subsequent neural policy will be robust or perform better. However, failing the test is more informative. If RL cannot even learn a strong policy that lies within the heuristic-equivalent class, adding more parameters is unlikely to resolve the training defect.

Compact policies also facilitate diagnosis. The heuristic litmus test becomes particularly useful because reducing the policy class also makes the training process amenable to visualization and diagnosis. Normally, the objective optimized by RL—the expected return—is intractable to evaluate over a high-dimensional parameter space (e.g., that induced by a deep neural policy). As a result, neither the return landscape nor the optimizer's trajectory through it is generally available for diagnosis. However, with a compact policy containing far fewer parameters, we can visualize the entire return landscape and relate its different regions to specific policy behaviors (example in Fig. 2a).

We realize this idea through three diagnostic views (§3.2): the return landscape, the optimization path across that landscape, and domain-specific correctness properties of the learned policy. Together, these views reveal failures that would otherwise appear as generic RL instability. For example, the return landscape may contain a desirable heuristic-quality point, yet poor initialization, high gradient variance, or excessively large policy updates may prevent the optimizer from reaching it. Conversely, optimization may behave exactly as intended, while a misweighted reward or an unrepresentative trace set reshapes the return landscape so that it favors aggressive or conservative behavior. Distinguishing between these failure modes provides actionable guidance on how to repair the training pipeline.

Case studies and findings. We first instantiate the heuristic litmus test from the ground up in a controlled ABR setting, which provides the most intuitive visualizations (§4). We find that even vanilla policy gradient reliably learns correct, near-optimal policies across a broad hyperparameter range. We also identify two representative failure modes: *unstable optimization paths* (sound landscape, straying optimizer) and *distorted return landscapes* (inappropriate reward weight or trace distribution shifts the optimum itself). Finally, we show that our methodology generalizes beyond the controlled ABR setting, including RL-based congestion control (§5).

Despite these encouraging results, our evaluation remains preliminary. In particular, our results do not yet explain training behavior in a full neural-network parameter space, which is notoriously difficult to study. Section 6 discusses more limitations and directions for future work.

The paper makes three contributions:

- We introduce *heuristic rediscovery* as a minimum capability test for RL in networked systems. When the policy class can represent a strong heuristic, training should learn a policy that matches its behavior and performance.
- We turn the compact policy class into a diagnostic instrument that combines three views that are normally hidden: the return landscape, the optimization path, and the correctness properties of the learned policy.
- We study the test primarily in ABR and reveal when and why training succeeds or fails. We further demonstrate that the methodology generalizes to congestion control.

2 BACKGROUND

RL and policy gradients. An RL policy $\pi_\theta(a \mid s)$ defines a probability distribution over actions a given state s, parameterized by θ: at each step t, the policy observes a state s_t, selects an action a_t, and receives a reward r_t; the resulting interaction sequence with the environment is called a trajectory. RL seeks parameters that maximize the *expected return*

$J(\theta)$, i.e., the accumulated reward averaged over trajectories drawn from π_θ (denoted $\mathbb{E}_{\pi_\theta}$):

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} r_t\right]. \tag{1}$$

Policy gradient methods [24] update the parameters by gradient ascent, $\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta J(\theta_k)$ with learning rate η, where the *policy gradient* $\nabla_\theta J(\theta)$ admits the analytical form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot \hat{R}_t\right], \tag{2}$$

where $\hat{R}_t$ is the reward-to-go, i.e., the sum of rewards from step t onward. Intuitively, ascending this gradient pushes up the (log) probabilities of actions that yield higher return.

Evaluating either expectation entails enumerating *all* possible trajectories, which is generally intractable (except under specific conditions we explain later); the policy gradient is therefore typically estimated with the sample mean of a batch of trajectories collected by the current π_θ. Vanilla Policy Gradient (VPG) [24] implements this approach and is what we use in our experiments.

Primary case: adaptive bitrate (ABR) streaming. An ABR client selects among a handful of pre-encoded bitrate levels for each few-second video chunk, trading video quality against rebuffering (playback stalls) to maximize the user's quality of experience (QoE). The widely deployed buffer-based algorithm (BBA) [11] makes this choice (almost) linearly in the playback buffer level—a policy class we return to in §3.2. Cast as RL, each chunk is one step: the state is the playback buffer level, the action a bitrate index, and the reward the chunk's QoE contribution; a trajectory is one video session. Pensieve [16] uses this formulation, training a deep neural network policy over manually selected inputs with an actor-critic variant of policy gradient [19]. Later studies found its behavior erratic and hard to audit [6, 7, 18, 25], and the Puffer deployment [27] reports that it does not consistently outperform the far simpler BBA.

Second case: congestion control (CC). A congestion controller adjusts a sender's rate to keep the bottleneck link utilized without building up queuing delay or packet loss. Here too, strong deployed algorithms are simple rules with a few parameters: delay-based schemes such as Vegas [4] and Copa [3] back off as queuing delay rises. Aurora [14] casts rate control as RL, training a deep neural policy on network statistics. §5.2 instantiates our methodology on CC.

3 HEURISTIC REDISCOVERY AS A LITMUS TEST

We propose heuristic rediscovery as a litmus test for diagnosing RL training. We first define the test and explain why it is a meaningful minimum capability check for an RL training pipeline (§3.1). We then introduce three diagnostic views that the test surfaces (§3.2), using Fig. 2 as a running example.

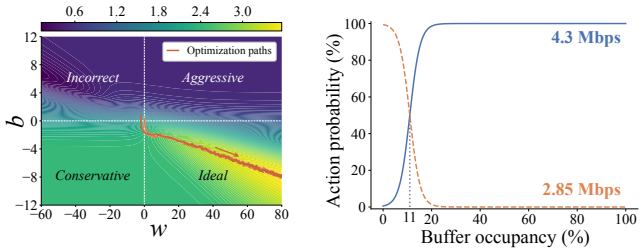


Figure 2: Diagnostic views rendered in the controlled 2-bitrate ABR setting. **(a)** Landscape of $J(\theta)$ over the plotted 2D parameter space, overlaid with optimization paths (V1 + V2). **(b)** Bitrate selection probability of the learned policy (V3).

3.1 The Litmus Test

Constructing the test. Many strong heuristics in networked systems encode their defining behavior with only a few parameters. For example, BBA and BOLA select higher bitrates as buffer occupancy grows [11, 23], while delay-based congestion-control algorithms such as Vegas and Copa [3, 4] reduce sending aggressiveness as queuing delay rises. These monotone rules can be represented by a compact policy class for RL to learn, such as logistic or softmax regression. Such a policy maps its observations linearly to an action logit in the binary case, or to one logit per action in the multi-action case. It then converts the logits into action probabilities using a sigmoid or softmax function. With no hidden layers and only a small set of weights and biases, this policy class is compact but expressive enough to represent the heuristic's defining input-to-action rule. We therefore restrict RL to this class and train it using reward alone. In the strict form of the test, the RL policy uses the same observations, actions, and decision interval as the heuristic, ensuring that failure reflects the training pipeline.

Requirements for passing. The policy must meet three requirements. First, *correctness*: it must satisfy the expected behavior of the heuristic. For buffer-based ABR, the bitrate must not decrease as the buffer grows. Thus, for $b_1 < b_2$,

$$\arg\max_a \pi_\theta(a \mid b_1) \le \arg\max_a \pi_\theta(a \mid b_2).$$

Second, *performance*: the policy must achieve utility within a stated margin of the heuristic. Third, *reliability*: correctness and performance must hold across a reasonable set of seeds, initializations, and training configurations. These requirements must be set before training, and the policy must meet all three to pass. Rediscovery does not require the same parameters or the same action in every state. It requires the same core control behavior and comparable utility.

Why this is a minimum capability check. The test has a known, attainable target. If the policy class can express the heuristic, a failed run cannot be blamed on insufficient model capacity. The cause must instead lie in the reward, training distribution, optimization, or implementation. Passing clears only this minimum bar; it does not show that a larger neural policy will be robust or better. Therefore, failing this test is more informative than passing it, and the views below help localize its cause.

3.2 Three Diagnostic Views

The requirements above determine whether training passes the test; the following views explain why. Because the policy class is compact, its expected return $J(\theta)$ can be readily estimated over the parameter space (computed exactly in the finite deterministic setting of §4). This yields three views that a reward curve alone cannot provide:

(V1) *Return landscape characterization.* V1 asks whether the reward and training environment make the required behavior high-return. The location and geometry of high-return regions—including ridges, basins, and plateaus—show which policies the objective favors and where gradient ascent can plausibly arrive [8]. Because V1 does not run an optimizer, it can expose a misaligned reward or trace distribution before training.

(V2) *Optimization path tracing.* V2 asks whether training follows the objective. Overlaying the parameter trajectory on V1 reveals whether the optimizer climbs a useful ridge, stalls on a plateau, or is deflected by gradient noise.

(V3) *Correctness properties.* V3 asks whether the learned policy satisfies the behavioral requirement specified by the test. We evaluate the action selected by the deployed policy rather than infer correctness from return: a policy can score well on the training objective while violating a domain requirement.

Example. Fig. 2 shows all three views for the controlled ABR setting in §4. The policy has two parameters, w and b, and selects the high bitrate with probability $\pi_\theta(\text{high}) = \sigma(w \cdot \text{buffer} + b)$. V1 plots the return landscape $J(w, b)$, V2 overlays the training paths, and V3 shows the resulting bitrate-selection rule. Together, these views divide the plotted parameter domain into four behavior classes: BBA-like (ideal), conservative, aggressive, and bitrate decreasing with buffer (incorrect). The last class violates the behavioral requirement even if it achieves high return.

4 A CONTROLLED ABR SETTING

In this section, we first trade realism for a training process that can be visualized and explained most effectively, by restricting the setting to two bitrates, short episodes, and constant-bandwidth traces. The resulting policy has two parameters but still represents a BBA-like buffer threshold [11], so we can plot its full return landscape and training paths without removing the target behavior. The short horizon makes every bitrate sequence enumerable, while constant traces isolate adaptation to capacity from temporal variation.

We emphasize that these are analytical conveniences, *not* requirements of the method: §5.1 restores the original large action space. Next, we show a base configuration that passes the test, then make changes to isolate two failure modes.

Setup. We use the policy from Fig. 2, with bitrates of 2.85 and 4.3 Mbps. Each episode contains eight 4-second chunks and runs on one of seven constant-bandwidth traces from 3.0 to 6.0 Mbps in 0.5 Mbps increments. The reward follows Pensieve [16]: $\text{bitrate}/\text{max_bitrate} - \lambda \cdot \text{stall_time}/\text{chunk_length}$ with $\lambda = 4.3$, and omits the smoothness term to keep the reward interpretable. The finite number of possible sequences lets us compute the expected return and gradient exactly; training itself uses sampled policy-gradient updates.

4.1 When Training Succeeds

Our base configuration initializes the policy near the origin and uses policy gradient with batch size 16 and Adam [15] at learning rate 0.01.

The pipeline passes. We check the three requirements of §3.1 directly. *Correctness* (V3): the learned policy increases the probability of the high bitrate as the buffer grows (Fig. 2b), satisfying the monotonicity requirement and recovering BBA's threshold structure from reward alone. *Performance*: we compare against BBA*, an oracle over deterministic one-threshold policies. Such a policy selects 2.85 Mbps below a buffer threshold and 4.3 Mbps above it; BBA* picks the threshold that maximizes average QoE over the seven traces. It is therefore optimal within the BBA-like behavior that the litmus test asks RL to rediscover. The learned policy's performance nearly matches BBA* using the base configuration (Fig. 3's red solid bar). *Reliability*: correctness and performance hold across repeated runs with near-origin initializations, batch sizes 1–64, and learning rates 0.005–0.1; QoE only degrades at more extreme learning rates and under SGD (Fig. 3). These experiments use the training traces for evaluation; they test the training pipeline, not generalization to unseen or time-varying traces.

Why it passes (V1 and V2). The verdict above is behavioral; the remaining two views explain it. The return landscape (V1) in Fig. 2a has a high-return ridge in the ideal quadrant and lower-return plateaus elsewhere. The ridge flattens as return increases, but gradients also vanish on the inferior plateaus, so a small gradient—or a flat reward curve—cannot certify success; V1 separates the desired ridge from places where training stops prematurely. The optimization paths

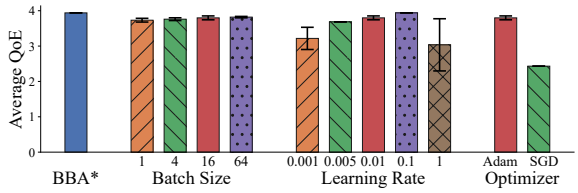


Figure 3: Average QoE across different configurations. Red solid bars represent the base configuration.

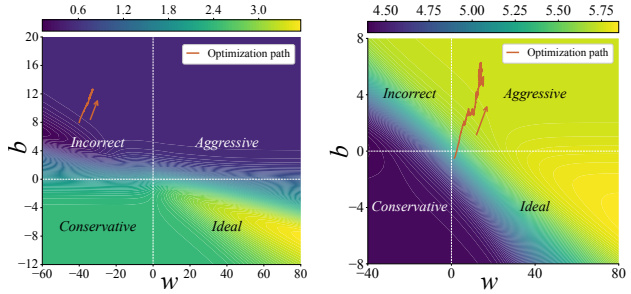


Figure 4: Controlled ABR exposes two failure modes. **(a)** F1: initialization at $(-40, 8)$ sends optimization away from the ideal ridge. **(b)** F2: $\lambda = 1.3$ shifts the high-return region toward aggressive policies.

(V2) overlaid in the same figure enter the ideal quadrant from near-origin initializations and follow the ridge toward higher return. The objective contains the target and training reaches it, so how it passes the test is explained.

This result does not extend to every initialization, optimizer, or larger neural policy. Those boundaries are the subject of the failures below.

4.2 When Training Fails

Outside the range tested above, two failure modes emerge, and the three diagnostic views (§3.2) cleanly separate them:

F1: unstable optimization paths. V1 still shows a high-return ridge in the ideal quadrant, but V2 misses it. For example, an initialization far from the origin makes the policy nearly deterministic, and the local gradient moves it farther into the flat, incorrect region instead of toward the ideal ridge (Fig. 4a). A large learning rate can instead overshoot the ridge and stop on a lower-return plateau; Fig. 3 shows the resulting QoE loss. In both cases, the target remains available without changing the reward or traces.

F2: distorted return landscapes. Changing the reward or traces changes V1. With $\lambda = 1.3$, the reward underweights stalls and moves the high-return region toward aggressive policies (Fig. 4b). Restricting training to 3.0–4.0 Mbps traces has the opposite effect: the high bitrate is rarely useful, so the landscape favors conservative policies. In both cases, the optimizer follows the objective it is given. Fixing the optimizer cannot restore the target; the reward or training distribution must change.

5 BEYOND THE CONTROLLED SETTING

The controlled ABR setting introduced earlier uses several simplifications to make the training process most amenable to visualization and diagnosis. We now relax them along two axes and ask whether the litmus test and its diagnostic views survive: restoring the full six-bitrate action space that Pensieve actually uses (§5.1), and applying the methodology on a different task, congestion control (§5.2).

5.1 ABR with Six Bitrates

We restore Pensieve's six-rate ladder, which spans 0.3–4.3 Mbps [16]. The policy assigns each bitrate a linear score, giving 12 parameters (w_i, b_i). Fixed-bandwidth traces extend to 0.5–6.0 Mbps so that the full ladder is usable (the lowest bitrate is 0.3 Mbps). The 12-D landscape cannot be exactly enumerated. We approximate V1 with 2-D projections and random-direction samples [2], and inspect V3 through action logits. These views expose the displayed failure, but they cannot rule out structure outside the sampled directions.

Success persists. We retain the controlled training configuration and add entropy regularization for early exploration in the larger action space [19]. The learned policies remain buffer-monotone. Because we cannot compute the optimal 12-parameter policy, we compare against BBA and BBA*. BBA uses Pensieve's default reservoir and cushion; BBA* generalizes the threshold oracle from §4 by tuning both parameters to maximize average QoE over the evaluation traces. In Fig. 5, RL Base uses the base configuration, whereas RL Tuned selects training parameters while holding the reward, traces, and policy class fixed. RL Base outperforms BBA, and RL Tuned approaches BBA*, across repeated runs and multiple training configurations.

Failure persists. Without entropy regularization, training converges to a conservative policy dominated by the lowest bitrates (Fig. 6b). Fig. 6a projects the return landscape onto the plane spanned by that policy and a BBA-like ideal policy. The actual path diverges from the ideal direction and reaches a different high-return region. This reproduces the failure signature from the controlled 2-bitrate setting: the objective makes undesirable behavior attractive.

5.2 Congestion Control

We next test whether the same diagnostic signatures appear in RL congestion control, which has a different state, action, and reward. The goal is to test whether the methodology transfers, not to design a new congestion-control algorithm.

Analyzable CC environment. Strong CC heuristics are also low-parameter, delay-reactive rules—TCP Vegas [4] and

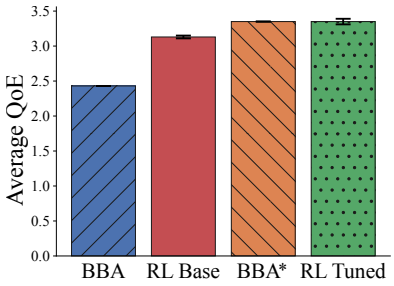


Figure 5: The base learned policy outperforms BBA; the tuned learned policy approaches BBA*.

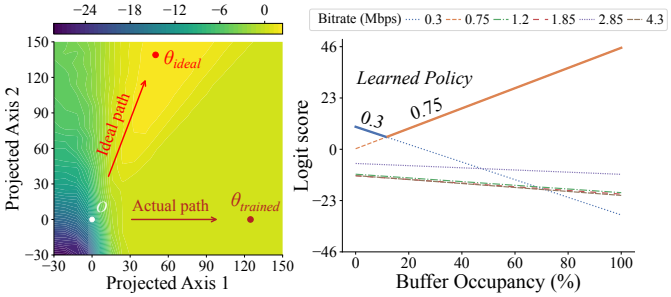


Figure 6: ABR six-bitrate failure mode. **(a)** On the plane with ideal and trained policies, the actual path does not reach the highest-return region. **(b)** The learned policy favors low-bitrate actions.

Copa [3] back off as queuing delay rises—so a compact policy on a delay signal is again the class an RL setup should be able to rediscover. Concretely, we use fixed-bandwidth bottlenecks (1–4 Mbps in 0.5 Mbps steps; 8 monitor intervals per session), a binary multiplicative rate action $a \in \{0.9, 1.1\}$, and a logistic-regression policy on one normalized signal—the RTT ratio $s = (\text{RTT}/\text{RTT}_{\text{min}}) - 1$. These together enable us to construct a 2-D space (w, b) where $J(\theta)$ is cheap to enumerate. RTT ratio plays buffer occupancy's role in ABR: the policy's own actions drive it within an episode (filling or draining the bottleneck queue), so it stays informative even on fixed-bandwidth traces. The reward follows prior RL CC work [14]: $r = \text{throughput} - \alpha \cdot \text{loss} - \beta \cdot \text{delay}$, terms normalized to $[0, 1]$, with $\alpha = 0.2$, $\beta = 0.6$ by default. We use delay-based heuristics because their back-off rule fits an interval-based RL formulation. Event-driven TCP does not share that formulation, and it reacts to individual ACK events. We further discuss how to expand our method under such mismatches in §6.

Correctness property (V3). A correct policy must not send faster as the RTT ratio rises: for $s_1 < s_2$, $\arg\max_a \pi_\theta(a \mid s_1) \geq \arg\max_a \pi_\theta(a \mid s_2)$. This reverses the ABR monotonicity check because high RTT is undesirable. The Ideal quadrant is therefore $w < 0, b > 0$; the Incorrect quadrant is $w > 0, b < 0$, where the policy sends faster as the queue grows.

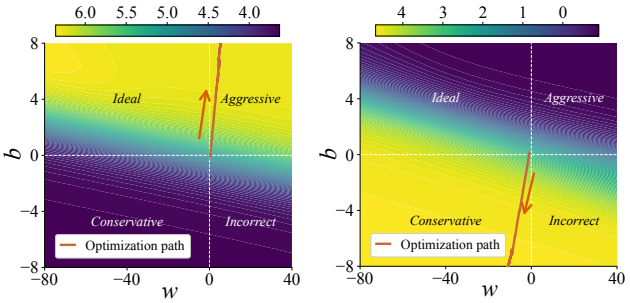


Figure 7: F2 recurs in congestion control. **(a)** A low delay penalty moves the ridge into the aggressive quadrant. **(b)** A narrow trace set favors a conservative policy.

The same signatures recur. Under the default configuration, V1 places the ridge in the Ideal quadrant and V2 reaches it. F1 recurs when an initialization in the Incorrect quadrant traps training there. F2 recurs in two ways: reducing the delay penalty to $\beta = 0.03$ moves the ridge into the Aggressive quadrant (Fig. 7a), while training only on 1–2 Mbps bottlenecks moves training into a conservative, low-throughput region (Fig. 7b). These match the signatures in ABR §4.2, demonstrating that our methodology is generalizable.

6 DISCUSSION AND FUTURE WORK

Validating the litmus test in full training stacks. Our controlled experiments simplify both the environment and the policy so that every diagnostic view is exact. They therefore omit stochastic traces, long horizons, actor-critic training, and deep policies. A complementary validation should instead preserve an existing RL pipeline—its simulator, trace corpus, episode length, and trainer—and replace only the policy with a heuristic-equivalent class. This tests whether reward-only training can rediscover a deployed heuristic under the conditions practitioners use. Failure would expose a limitation of the full pipeline rather than an artifact of our controlled environment; recurrence of F1 and F2 under the same triggers would show that these diagnostic signatures transfer beyond the controlled setting.

For ABR, this experiment is direct. Pensieve and BBA solve the same control problem—one decision per chunk, the buffer as a shared input, a bitrate as the action—so the reduction only shrinks the policy's input and size. The challenges are methodological. $J(\theta)$ is no longer enumerable, so V1 must rely on projections and sampled directions [2]. Returns are stochastic, so every estimate needs more trajectories. Correctness checks must extend beyond single-input monotonicity once the policy reads more features, and the diagnosis becomes behavioral, resting largely on V3. How much diagnostic power survives these approximations is the question future work must answer.

When the heuristic and RL disagree. The CC experiment illustrates the mismatch described in §5.2: TCP and RL-based CC do not solve the same control problem, in three ways. (1) *Time step:* networks deliver events faster than an RL policy can act, so practical RL controllers decide over fixed monitor intervals [14], whereas TCP reacts to individual events—each ACK, triple duplicate ACK, or timeout [13]. The RL formulation deviates from the heuristic before training starts. (2) *State:* RL summarizes each interval into statistics—Aurora observes latency ratios, latency gradients, and sending ratios—whereas the heuristic's effective state is the event type itself. (3) *Action:* TCP adjusts a congestion window in an AIMD manner, whereas Aurora adjusts the sending rate.

We see two ways to reconcile the mismatch. *Bring RL closer to the heuristic*: constrain the RL controller to a TCP-compatible, event-driven state and action space, so that TCP itself becomes the rediscovery target. *Bring the heuristic to RL*: design new strong heuristics—perhaps with LLMs [10]—that operate on RL's time step and state-action space, giving the test a target there. Our CC experiments (§5.2) took the second route implicitly, by targeting delay-based heuristics whose back-off rule fits an interval formulation.

Conclusion. Heuristic rediscovery gives RL training in networked systems a minimum capability check with a known, attainable target, and the compact policy class behind it makes training inspectable: the objective, the optimization path, and the learned behavior. In controlled ABR and CC settings, these views separate optimization failures from reward- and trace-induced objective failures. The path forward is what Fig. 1 draws: validate the test in full training stacks, construct it for heuristics that do not share RL's formulation, and let RL earn complexity step by step.

REFERENCES

[1] Neil Agarwal, Rui Pan, Francis Y. Yan, and Ravi Netravali. 2025. Mowgli: Passively Learned Rate Control for Real-Time Video. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '25)*.

[2] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. 2019. Understanding the Impact of Entropy on Policy Optimization. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR.

[3] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*.

[4] L.S. Brakmo and L.L. Peterson. 1995. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* (1995), 1465–1480.

[5] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. 2018. Cellular Network Traffic Scheduling with Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '18)*.

[6] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. 2021. Verifying Learning-augmented Systems. In *Proceedings of the ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA.

[7] Tomer Gilad, Nathan H. Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. 2019. Robustifying Network Protocols with Adversarial Examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*.

[8] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. 2014. Qualitatively Characterizing Neural Network Optimization Problems. *arXiv preprint arXiv:1412.6544* (2014).

[9] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[10] Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y. Yan. 2024. Designing Network Algorithms via Large Language Models. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets '24)*. Association for Computing Machinery, New York, NY, USA.

[11] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA.

[12] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. 2022. AI/ML for Network Security: The Emperor has No Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*.

[13] V. Jacobson. 1988. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols* (Stanford, California, USA) *(SIGCOMM '88)*. Association for Computing Machinery, New York, NY, USA, 314–329.

[14] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 3050–3059.

[15] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA.

[17] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA.

[18] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting deep learning-based networking systems. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA.

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*. PMLR.

[20] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose. 2000. Modeling TCP Reno Performance: a Simple Model and its Empirical Validation. *IEEE/ACM Transactions on Networking* 8, 2 (2000), 133–145.

[21] Lorenzo Pappone, Alessio Sacco, and Flavio Esposito. 2025. Mutant: Learning Congestion Control from Existing Protocols via Online Reinforcement Learning. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '25)*.

[22] Sagar Patel, Dongsu Han, Nina Narodytska, and Sangeetha Abdu Jyothi. 2025. Agua: A Concept-Based Explainer for Learning-Enabled Systems. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA.

[23] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711.

[24] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems* 12 (1999).

[25] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. 2022. Genet: Automatic Curriculum Generation for Learning Adaptation in Networking. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA.

[26] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. 2023. Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA.

[27] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in Situ: a Randomized Experiment in Video Streaming. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*.

[28] Mohammad Zangooei, Jannis Weil, Amr Rizk, Mina Tahmasbi Arashloo, and Raouf Boutaba. 2026. Analyzing Symbolic Properties for DRL Agents in Systems and Networking. *Proc. ACM Meas. Anal. Comput. Syst.* 10, 2, Article 29 (2026).

[29] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. 2020. OnRL: Improving Mobile Video Telephony via Online Reinforcement Learning. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*.