# Dynamic Control of Queuing Networks via Differentiable Discrete-Event Simulation

**Ethan Che** [*1] **Jing Dong** [1] **Hongseok Namkoong** [1]

## Abstract

Queuing network control is a problem that arises in many applications such as manufacturing, communications networks, call centers, hospital systems, etc. Reinforcement Learning (RL) offers a broad set of tools for training controllers for general queuing networks, but standard model-free approaches suffer from high variance of trajectories, large state and action spaces, and instability. In this work, we develop a modeling framework for queuing networks based on discrete-event simulation. This model allows us to leverage tools from the gradient estimation literature to compute approximate *first-order* gradients of sample-path performance metrics through auto-differentiation, despite discrete dynamics of the system. Using this framework, we derive gradient-based RL algorithms for policy optimization and planning. We observe that these methods improve sample efficiency, stabilize the system even when starting from a random initialization, and are capable of handling non-stationary, large-scale instances.

## 1. Introduction

Queuing networks are ubiquitous for modeling job-processing systems, which arise in applications related to manufacturing systems (Perkins & Kumar, 1989), call centers (Bassamboo et al., 2006), communications networks (Maguluri et al., 2012), hospitals (Dai & Shi, 2019), and more. In these systems, jobs arrive to queues at random interarrival epochs and are either routed to a server to be processed, or wait in the queue if no servers are available. After a random processing time, they either leave the system or are routed to further servers for further processing. The optimal control problem is determining how to dynamically allocate server capacity to jobs in order to optimize various performance metrics, such as minimizing average waiting time of jobs in the queue, average length of queues, maximizing job-server compatibility, etc.

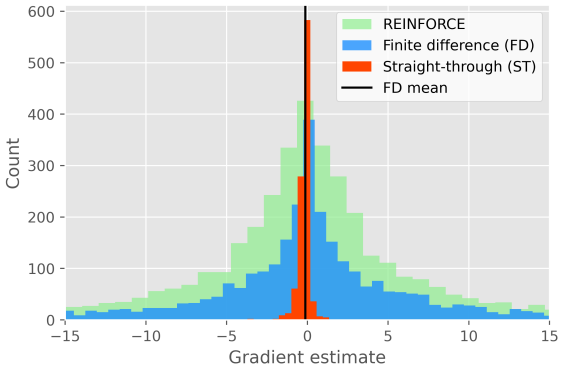


*Figure 1.* Histogram of gradient estimators for average queue length of a linear softmax policy in the N-model network (see Appendix A.2), across a horizon of $N = 1000$ steps.

Solving this control problem is known to be difficult, due to high stochasticity, nonlinear and non-smooth dynamics, non-stationarity, and complexity of the network topology. While there are many effective heuristics such as the $c\mu$-rule (Mandelbaum & Stolyar, 2004), the MaxWeight policy (Dai & Lin, 2005), or fluid policies (Chen & Yao, 1993), their performance can vary across different problem settings, and theoretical guarantees are only available for specific network architectures and specific load regimes. For these reasons, recently there has been much interest in Reinforcement Learning (RL) as a way to develop control policies for generic queuing networks. While there has been much development in applying model-free RL, these approaches face challenges due to high variance of trajectories, large state and action spaces, as well as instability of the system under poor controllers (Meyn, 2008; Dai & Gluzman, 2022; Liu et al., 2022; Zaki et al., 2021).

In this work, we provide a model-based framework for policy optimization and planning for controlling multi-class, multi-pool queuing networks through the lens of discrete event simulation. Rather than standard RL approaches for queuing, which typically rely on a discrete-time Markov De-

[*]Equal contribution [1]Decision, Risk and Operations, Columbia Business School, New York, US. Correspondence to: Ethan Che <eche25@gsb.columbia.edu>.

*Published at the Differentiable Almost Everything Workshop of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. July 2023. 

cision Process (MDP) representation which only holds when arrivals and service times are stationary and are exponentially distributed, we model the system as a discrete event dynamical system (Ho, 1987; Ho & Cao, 2012; Tsitsiklis, 1989) in continuous time, sampling the system whenever an arrival or service occurs. This model only relies on samples of event times and allows for non-stationary and non-Markovian arrival and service processes, which better matches behavior of real-world systems (Green et al., 2007).

Importantly, this modeling framework provides a novel way to obtain approximate first-order derivatives of performence metrics with respect to controls applied to the system. First, we use a continuous relaxation of the action space motivated by fluid models to obtain a 'reparameterization trick' (Kingma & Welling, 2013) of the dynamics with respect to the actions. While these dynamics are non-differentiable, we apply the 'straight-through' trick (Bengio et al., 2013) to obtain gradients. This way, we can directly compute the gradient of rewards or costs accumulated in a rollout through reverse-mode autodifferentiation. This can be done in parallel with minibatches of rollouts using GPUs. While straight-through gradients are known to be potentially biased, we observe the bias is very small in practice even for long episodes, and the variance is significantly lower than standard zero-order estimators (see Figure 1 and Figure 2).

We propose two RL algorithms based on these first-order gradients. *Straight-through Policy Gradient* (StraightPG) updates policy parameters using straight-through gradients of cost computed via autodifferentiation along mini-batches of sample paths. *Straight-through Planning* is a model-predictive control style policy that solves a planning problem via straight-through gradients. Our preliminary results show that despite relying on approximate gradients, these policies match the performance of well-established heuristics and model-free RL algorithms on standard instances, but with improved sample efficiency. And unlike other RL methods, which require switching to a stabilizing policy (Liu et al., 2022; Zaki et al., 2021) or behavorial cloning of a stabilizing policy (Dai & Gluzman, 2022), the training process is robust to instability even when starting from a random initialization.

## 2. Model

We consider a continuous-time multi-class, multi-pool queuing network with $m$ servers and $n$ queues corresponding to $n$ different job types, with time indexed by $t \in \mathbb{R}_+^n$. The topology of the queuing network $M = \{0,1\}^{m\times n}$ describes the compatibility of servers and jobs, with $M_{ij} = 1$ if server $i$ can serve job type $j$. The job lengths (or state) $x(t) \in \mathbb{N}^n$ are the number of jobs of each type still remaining in the system, including jobs waiting in queue and jobs in service. Jobs arrive according to stochastic interarrival times

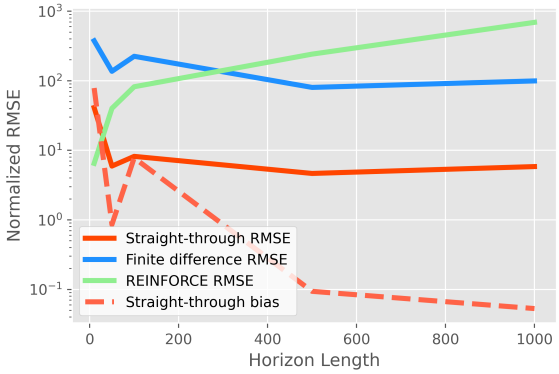


*Figure 2.* Normalized RMSE and bias of gradient estimators for average queue length of a linear softmax policy in the N-model network (see Appendix A.2) across a horizons ranging from $N = 50$ to $N = 1000$ steps. Normalized by the magnitude of the average gradients.

$\tau^A(t) \in \mathbb{R}_+^n$, which may be depend on time $t$, and if they are not served immediately they wait in queue.

### 2.1. Continuous action relaxation

Upon observing the state, the controller chooses an action $a \in P := \{X \in \mathbb{R}_+^{m\times n} : X1 = 1, 1^\top X = 1\}$, which is an assignment of job types to servers, which must be feasible according to $M$. If server $i$ is assigned to job type $j$, then a job from queue $j$ is routed first-in first-out (FIFO) to the server, upon which the server begins processing it. The processing time for each job-server pair is described by $\tau^S(t) \in \mathbb{R}_+^{m\times n}$, which may different across job-server pairs and can vary across time. Yet, since the servers only process jobs routed to them according to $a$, the service time for server $i$ and a job of type $j$ is equal to $\tau_{ij}^S$ only if $a_{ij} = 1$, and infinite otherwise.

Drawing inspiration from fluid models (Chen & Yao, 1993), we consider a novel continuous relaxation of the dynamics, in which the realized service times are equal to $\tau^S/a$, with division being element-wise. This allows for non-integral assignments, which represent a server splitting its capacity between several jobs or a job being served by several servers. This way, we consider *deterministic* policies over a *continuous* relaxation of the action space, which departs from standard RL treatments of queuing control that typically use stochastic policies over discrete actions. Note that we use this relaxation for training and optimization, but for evaluation we restrict to integral assignments. After a job is served it leaves the system or enters another queue.

### 2.2. Discrete event dynamical system

Although the process evolves in continuous time, we can obtain a discrete-time characterization by sampling the process when an arrival or a service occurs. We refer to this

as the *discrete-event dynamical system* model. Let $\tau_k \in \mathbb{R}$ for $k \in \mathbb{N}_+$ be denote as the $k$th *event epoch*. We define $x_k := x(\tau_k)$ to be the job length process sampled at $\tau_k$, and let $\tau_k^A, \tau_k^S$ be the arrival times and service times at epoch $k$. Starting from $\tau_0 = 0$, the next event epoch is determined as the minimum of the all the total possible event times: $\tau_{k+1} = \min\{\tau_k^A, \tau_k^S / a\}$.

Define $e(\tau_k^A, \tau_k^S/a) \in \{0,1\}^{2n}$ as the one-hot vector denoting which event had the minimum time. The first $n$ entries refer to an arrival for type $j$, the latter $n$ entries refer to service of type $j$. The model is also described by an event matrix $\Delta \in \mathbb{R}^{2n \times n}$, where each row $i$ corresponds to how the job lengths $x(t)$ are updated if the $i$th event is the minimum event time. With this, the dynamics are:

$$\tau_{k+1} = \min\{\tau_k^A, \tau_k^S/a\} \tag{1}$$

$$x_{k+1} = x_k + \Delta^\top e(\tau_k^A, \tau_k^S/a) \tag{2}$$

where the first equation describes the event epoch and the second equation is the update for the state. The arrival and service times of other jobs are reduced by $\tau_{k+1}$ and are reset if the event was the minimum time event.

**Example 2.1.** Consider a single server $M/M/1$ queue with arrival rate $\lambda$ and service rate $\mu$. Interarrivals are $\tau_A \sim Exp(1/\lambda)$, services are $\tau_S \sim Exp(1/\mu)$, $a = \mathbf{1}\{x > 0\}$,

$$\Delta^\top = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, e(\tau^A, \tau^S/a) = \begin{cases} (1,0) & \tau^A < \tau^S/a \\ (0,1) & \tau^A > \tau^S/a \end{cases}$$

This can be seen as a 'reparameterization trick' (Kingma & Welling, 2013) of the dynamics with respect to actions; enabling us to evaluate different actions under the same sample path of stochastic inputs, i.e. $\tau^A$ and $\tau^S$.

### 2.3. Control problem

The objective of the controller is to minimize an average cost metric, such as the average job lengths or the average time the system is busy. Let $c(x,a)$ be the instantaneous cost given job lengths $x$ and assignment $a$. A typical cost is a $c(x,a) = h^\top x$ where $h$ is a vector of holding costs associated with each job type. The objective is to minimize the average cost over a horizon of length $T$: $\min \int_0^T c(x(t), a(t))dt$. In the discrete-event dynamical system, we replace this with the average of the cost over the event times: $\min \sum_{k=1}^{N_T} c(x_k, a_k)(\tau_{k+1} - \tau_k)$. where $N_T$ is the number of events that occur before time $T$.

### 2.4. Straight-through gradients

The key source of non-differentiability in the model is $e(\tau_k^A, \tau_k^S/a)$, which is a onehot argmin of the event times. If this could replaced with a differentiable surrogate, the job-lengths would evolve as a differentiable function of the action $a$. To do so, we employ the straight-through trick to differentiate through $e$, a trick which has been used effectively in discrete representation learning (Van Den Oord et al., 2017) and model-based RL (Hafner et al., 2020). In other words, we replace $e$ with

$$\hat{e}(\tau_k^A, \tau_k^S/a) := e(\tau_k^A, \tau_k^S/a) + \text{softmin}(\tau_k^A, \tau_k^S/a) - \text{sg}[\text{softmin}(\tau_k^A, \tau_k^S/a)] \tag{3}$$

where $\text{sg}[\cdot]$ is the stop gradient operator. This will produce identical trajectories as in the update (1) in the forward pass, but behaves as a softmin in the backward pass.

As a result, we can estimate gradients of the cost of a trajetory with respect to any routing action or sequence of actions taken along the way: $\nabla_{a_k} \sum_{k=1}^N c(x_k, a_k)(\tau_{k+1} - \tau_k)$. Figure 1 and 2 compare the variance and bias of the straight-through estimator with finite differences (using the continuous relaxation and under coupled random seeds) and the REINFORCE estimator for a simple linear policy. We observe that the estimator achieves significant variance reduction over alternatives, with low bias.

**Algorithm 1** Straight-through Policy Gradient (StraightPG)

**Input:** Initial state $x_0$, learning rate $\alpha$, epochs $m$, episode length $N$, initial policy $\rho_{\theta_0}$
**for** $i = 0$ **to** $m-1$ **do**
  Draw a sample path and compute cumulative cost $V(x_0) = \sum_{k=1}^N c(x_k, a(\rho_{\theta_i}(x_k)))(\tau_{k+1} - \tau_k)$.
  Compute $\nabla_{\theta_i} V(x_0)$ with straight-through trick.
  Update $\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_{\theta_i} V(x_0)$.
**end for**

### 2.5. Sinkhorn parameterization

Instead of optimizing over actions $a \in P$, we parameterize actions in terms of priorities $\rho \in \mathbb{R}^{m \times n}$ which produce assignments through linear assignment: $\bar{a}(\rho) = \max_{a \in P} \text{Tr}(a\rho)$.

**Example 2.2.** The $c\mu$-rule uses a static priority $\rho_{ij} = h_j \mu_{ij}$ where $\mu_{ij}$ is service rate of server $i$ and job type $j$. MaxWeight uses a state-dependent priority $\rho_{ij} = x_j h_j \mu_{ij}$.

To maintain differentiability, we replace the hard linear assignment with Sinkhorn's algorithm: $a(\rho) = \max_{a \in P} \text{Tr}(a\rho) + \lambda \text{KL}(a | 11^\top)$. Thus, actions only specify relative values of assigning a server to a job, rather than specifying the assignment exactly.

## 3. Algorithms and Empirical Results

Using the first-order gradients obtained through the straight-through trick and Sinkhorn parameterization, we propose two policies. First, *Straight-through Policy Gradient*, which

**Algorithm 2** Straight-through Planning (StraightPlan)

**Input:** Initial state $x_0$, total episode length $N$, planning horizon $H$.
**for** $k = 0$ **to** $N$ **do**
  Solve planning problem $\min_{\rho_{k:k+H}} V(x_k) = \sum_{i=k}^{k+H} c(x_i, a(\rho_i))(\tau_{k+1} - \tau_k)$ via gradient descent on sample-path rollouts.
  Play action $a_k = \bar{a}(\rho_k)$ via linear assignment.
  Update states and times $(x_{k+1}, \tau_{k+1})$ with (1).
**end for**

parameterizes priorities $\rho_\theta$ that produce actions through Sinkhorn's algorithm. The parameter $\theta$ is updated by gradients of the cost from sample rollouts directly through auto differentiation. Second, *Straight-through Planning*, which is a model-predictive control (MPC) style policy that solves a planning problem via gradients of sample-path rollouts. We evaluate the performance of the policy on a few examples.

### 3.1. Criss-cross Network and Reentrant Network

To further evaluate the quality of the straight-through gradients, we train policy gradient to minimize the stationary average queue-length of the criss-cross network (Harrison & Wein, 1990), which is a simple tandem queue with 2 servers and 3 job types. We evaluate across different load levels (see Table 4 in Appendix A.3). To assess validity for large systems, we train the method on a class of reentrant queuing networks introduced in (Bertsimas et al., 2014) across different system sizes. We compare against benchmarks such as the robust fluid policy (RFP) (Bertsimas et al., 2014) and Proximal Policy Optimization (PPO) as implemented in (Dai & Gluzman, 2022), which uses several variance reduction strategies. We observe in Table 1 and 2 that despite potential bias in the gradients, straight-through policy gradient outperforms RFP and achieves similar performance to PPO, typically using fewer policy iterations and without requiring an initial behavior cloning step to ensure stability.

### 3.2. Parallel-server system

To test the applicability of the model to non-stationary settings, we consider a parallel-server system with 5 servers and 5 queues (Chen et al., 2021). We consider a non-stationary arrival process where queues 1 and 3 experience a demand surge until time $t = 100$. We compare against $c\mu$-rule, MaxWeight, and the fluid policy in Table 3. We observe that the gradient-based algorithms outperform other policies and do so by planning ahead and tapering service to the overloaded queues before the demand surge ends.

*Table 1.* Average Queue Length in Criss Cross Network

| SETTING | OPTIMAL | RFP | PPO | STRTPG |
|---|---|---|---|---|
| IMB.LIGHT | 0.671 | 0.677 | 0.671 | 0.673 |
| BAL.LIGHT | 0.843 | 0.855 | 0.844 | 0.854 |
| IMB.MED | 2.084 | 2.133 | 2.084 | 2.107 |
| BAL.MED | 2.829 | 2.920 | 2.833 | 2.888 |
| IMB.HEAVY | 9.97 | 10.10 | 10.01 | 10.01 |
| BAL.HEAVY | 15.23 | 15.58 | 15.35 | 15.34 |

*Table 2.* Average Queue Length in Reentrant Network

| NUM CLASSES | RFP | PPO | STRTPG |
|---|---|---|---|
| 6 | 15.42 | 14.13 | 15.68 |
| 9 | 24.92 | 23.27 | 22.43 |
| 12 | 36.86 | 32.17 | 32.38 |
| 15 | 43.63 | 39.30 | 37.51 |
| 18 | 52.98 | 51.47 | 48.36 |
| 21 | 59.05 | 55.12 | 55.45 |

*Table 3.* Average Holding Cost in Parallel Server System

| POLICY | AVG HOLDING COST |
|---|---|
| $c\mu$-RULE | 107.01 |
| FLUID | 83.35 |
| MAXWEIGHT | 88.34 |
| STRAIGHTPG | 77.29 |
| STRAIGHTPLAN | 78.25 |

## 4. Related Work

Our work directly builds on approaches for queuing in RL as in (Zaki et al., 2021; Dai & Gluzman, 2022; Liu et al., 2022), and proposes a new gradient estimator. Our approach is related to differentiable simulation for policy optimization (Suh et al., 2022; Mora et al., 2021), as it uses exact gradients to update policies. Our work builds on previous literature on sensitivity analysis in discrete event systems (Ho & Cao, 2012; Ho, 1987), but methodologically approaches this from the viewpoint of gradient estimation for discrete random variables (Tucker et al., 2017; Mohamed et al., 2020). Of particular relevance is (Arya et al., 2022), who propose a framework for autodifferentation (AD) with discrete random variables, which produces unbiased gradient estimates with stochastic coupling for variance reduction. This offers a promising method for queuing network control. Our approach instead constructs a differentiable 'reparameterization trick' of the dynamics with respect to actions for reverse-mode AD, appropriate for computing gradients of scalar costs with respect to high-dimensional policy parameters and actions. While their framework also allows for reverse-mode AD, gradients are no longer unbiased due to non-linear dynamics. Our estimator has low bias in practice and significantly reduced variance than the finite-difference estimator with coupled random seeds (in Figure 1 and 2), which suggests that embedding discrete actions into a continuous space and reparameterizing the dynamics may obtain greater variance reduction than stochastic coupling alone.

## References


Arya, G., Schauer, M., Schäfer, F., and Rackauckas, C. Automatic differentiation of programs with discrete randomness. *Advances in Neural Information Processing Systems*, 35:10435–10447, 2022.

Bassamboo, A., Harrison, J. M., and Zeevi, A. Design and control of a large call center: Asymptotic analysis of an lp-based method. *Operations Research*, 54(3):419–435, 2006.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bertsimas, D., Nasrabadi, E., and Paschalidis, I. C. Robust fluid processing networks. *IEEE Transactions on Automatic Control*, 60(3):715–728, 2014.

Chen, H. and Yao, D. D. Dynamic scheduling of a multiclass fluid network. *Operations Research*, 41(6):1104–1115, 1993.

Chen, J., Dong, J., and Shi, P. Optimal routing under demand surges: The value of future arrival rates. *Available at SSRN 3980227*, 2021.

Dai, J. G. and Gluzman, M. Queueing network controls via deep reinforcement learning. *Stochastic Systems*, 12(1): 30–67, 2022.

Dai, J. G. and Lin, W. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53(2):197–218, 2005.

Dai, J. G. and Shi, P. Inpatient overflow: An approximate dynamic programming approach. *Manufacturing & Service Operations Management*, 21(4):894–911, 2019.

Green, L. V., Kolesar, P. J., and Whitt, W. Coping with time-varying demand when setting staffing requirements for a service system. *Production and Operations Management*, 16(1):13–39, 2007.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Harrison, J. M. and Wein, L. M. Scheduling networks of queues: Heavy traffic analysis of a two-station closed network. *Operations research*, 38(6):1052–1064, 1990.

Ho, Y.-C. Performance evaluation and perturbation analysis of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 32(7):563–572, 1987.

Ho, Y.-C. L. and Cao, X.-R. *Perturbation analysis of discrete event dynamic systems*, volume 145. Springer Science & Business Media, 2012.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Liu, B., Xie, Q., and Modiano, E. Rl-qn: A reinforcement learning framework for optimal control of queueing systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 7(1):1–35, 2022.

Maguluri, S. T., Srikant, R., and Ying, L. Stochastic models of load balancing and scheduling in cloud computing clusters. In *2012 Proceedings IEEE Infocom*, pp. 702–710. IEEE, 2012.

Mandelbaum, A. and Stolyar, A. L. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized c$\mu$-rule. *Operations Research*, 52(6): 836–855, 2004.

Meyn, S. *Control techniques for complex networks*. Cambridge University Press, 2008.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte carlo gradient estimation in machine learning. *The Journal of Machine Learning Research*, 21(1):5183–5244, 2020.

Mora, M. A. Z., Peychev, M., Ha, S., Vechev, M., and Coros, S. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pp. 7805–7817. PMLR, 2021.

Perkins, J. R. and Kumar, P. Stable, distributed, real-time scheduling of flexible manufacturing/assembly/diassembly systems. *IEEE Transactions on Automatic Control*, 34(2):139–148, 1989.

Suh, H. J., Simchowitz, M., Zhang, K., and Tedrake, R. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pp. 20668–20696. PMLR, 2022.

Tsitsiklis, J. N. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2 (2):95–107, 1989.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Zaki, M., Mohan, A., Gopalan, A., and Mannor, S. Improper reinforcement learning with gradient-based policy optimization. *arXiv preprint arXiv:2102.08201*, 2021.

# A. Appendix A

## A.1. Implementation

We detail the Straight-through Policy Gradient algorithm and the Straight-through planning policy in Algorithms 1 and 2 respectively.

For the criss-cross network and reentrant networks, we parameterize the policy $\rho_\theta$ as a multi-layer perceptron (MLP) with input dimension equal to the number of queues, output dimension $m \times n$ and 3 hidden layers with 128 hidden units in each layer. For the criss-cross network, each policy gradient step uses a mini-batch of 50 episodes in parallel of length $N = 5,000$ for light loaded, $N = 10,000$ for medium loaded systems, and $N = 20,000$ steps for heavily loaded system. For the reentrant network each policy gradient step uses a mini-batch of 50 episodes in parallel of length $N = 20,000$ steps.

For the parallel-server system, due to non-stationary, we also input time $t$ as another input variable, with all else being the same.

## A.2. N-model

The N-model is a network with two queues and two servers. Each queue has it's own designated server, but server 1 can also capable of serving jobs in queue 2, at a reduced service rate. The network topology is:

$$M = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

In Figure 1 and Figure 2, we consider a system where interarrivals of the queues are distributed $Exp(1/\lambda_1)$ and $Exp(1/\lambda_2)$ with $\lambda_1 = 0.4\rho$ and $\lambda_2 = 1.3\rho$ where $\rho = 0.95$. Service times of server i and queue j are distributed $Exp(1/\mu_{ij})$ with service rates $\mu_{ij}$:

$$\mu = \begin{pmatrix} 1 & 0.5 \\ 0 & 1 \end{pmatrix}$$

To assess the quality of gradients, we consider a linear softmax policy of the form:

$$a(x) = \text{softmax}\left(\begin{bmatrix} \theta_{11}x_1 & \theta_{12}x_2 \\ 0 & \theta_{22}x_2 \end{bmatrix}\right)$$

where the softmax is performed row-wise. We take gradients of the cumulative holding cost with respect to the parameters $\theta$, evaluated at $\theta_{11} = \theta_{12} = \theta_{22} = 2$ across an episode of $N$ events:

$$\nabla_\theta V_\theta(x_0) = \nabla_\theta \left[\sum_{k=1}^{N} x_{k,1} + x_{k,2}\right]$$

For the straight-through and finite difference estimators, we calculate the gradient with respect to the non-integral assignment $a(x)$. For the finite difference estimator, we use the two point estimator with $\sigma = 0.1$:

$$\nabla_\theta V_\theta(x_0) = \frac{1}{\sigma}\mathbb{E}\left[V_{\theta+\sigma Z}(x_0) - V_\theta(x_0)\right]$$

where $Z \sim N(0, I) \in \mathbb{R}^3$ is a standard normal random vector. For both rollouts, we use the same random seed, which couples the arrival and service times. Note that the finite difference estimator is using the same continuous action relaxation as the straight-through estimator and produces an unbiased estimate of $\nabla_\theta V_\theta(x_0)$. For this reason, we use this estimator to measure the bias of the straight-through gradients.

To compare the REINFORCE estimator, which typically considers stochastic policies over discrete actions, we look at a policy that instead of choosing the action $a(x)$, samples from the softmax to produce a hard assignment. Since the evaluated policy is different, the gradient differs from what we obtain from the straight through and finite difference estimators. While the sign of the REINFORCE gradients are the same, they tend to be off by a scale factor. For a comparison, in Figure 2 we present the normalized root mean squared error (RMSE) of the different gradient estimators, where the RMSE is normalized by the magnitude of the gradient. For the straight-through estimator, RMSE is calculated by taking the mean squared error with the average finite difference estimator (across 100,000 estimates). For the other estimators, RMSE is the square root of sum of variances of each component, considering both estimators to be unbiased. In Figure 1, we rescale the gradient so that its mean is equal to that of the finite-difference estimator.

*Table 4.* Load parameters in each settings

| SETTING | $\lambda_1$ | $\lambda_3$ | $\mu_{11}$ | $\mu_{13}$ | $\mu_{22}$ |
|---|---|---|---|---|---|
| IMB.LIGHT. | 0.3 | 0.3 | 2 | 1.5 | 2 |
| BAL.LIGHT. | 0.3 | 0.3 | 2 | 1 | 2 |
| IMB.MED. | 0.6 | 0.6 | 2 | 1.5 | 2 |
| BAL.MED. | 0.6 | 0.6 | 2 | 1 | 2 |
| IMB.HEAVY. | 0.9 | 0.9 | 2 | 1.5 | 2 |
| BAL.HEAVY. | 0.9 | 0.9 | 2 | 1 | 2 |

### A.3. Criss-cross

The criss-cross network (Harrison & Wein, 1990) is a network with 3 queues and 2 servers. Queues 1 and 3 can be served by server 1. When a queue 1 job is served, it joins queue 2 which is served by server 2. When jobs in queue 2 and 3 are served, they leave the system. The network topology is:

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

We consider a system where interarrivals of the queues are distributed $Exp(1/\lambda_1)$, $Exp(1/\lambda_2)$ with service times of server i and queue j are distributed $Exp(1/\mu_{ij})$ with service rates $\mu_{ij}$. Due to the tandem nature of the queue, the event matrix $\Delta$ is:

$$\Delta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

where the first 3 rows describe the state change when an arrival occurs (note that no arrivals to queue 2 occur unless they leave from queue 1) and the second 3 rows discribe the state change when a service occurs. We consider different parameter settings for the load on the system, described in Table 4.

### A.4. Reentrant network

The reentrant queuing network is parameterized by $L$, which is the number of layers. There are $L$ servers and $3L$ many job classes/queues. In each layer, there are 3 queues and each can be served by a single server, who only serves those 3 queues. Job types have an index $j$ ranging from $0, ..., 3L-1$. After a job of type $j$ is served, the job is sent to queue $i+3$. F or jobs of type $3L-3$, after the job is processed it is sent to job type 1. For jobs of type $3L-2$ and $3L-1$ after service they are immediately processed.

Arrivals only occur to queues 0 and 2. The arrival times for both queues are distributed $Exp(140/9)$. Servers are indexed by $i$ ranging from $0, ..., L$. Service times are exponential $Exp(1/\mu_{ij})$ with

$$\mu_{ij} = \begin{cases} 1/8 & i \bmod 2 = 0 \text{ and } j \bmod 3 = 0 \\ 1/2 & i \bmod 2 = 0 \text{ and } j \bmod 3 = 1 \\ 1/4 & i \bmod 2 = 0 \text{ and } j \bmod 3 = 2 \\ 1/6 & i \bmod 2 = 1 \text{ and } j \bmod 3 = 0 \\ 1/7 & i \bmod 2 = 1 \text{ and } j \bmod 3 = 1 \\ 1 & i \bmod 2 = 1 \text{ and } j \bmod 3 = 2 \end{cases}$$

### A.5. Parallel-server

We consider a parallel-server system with 5 servers and 5 queues. The network topology is:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The vector of holding costs on the job lengths for each queue is $[2, 1, 1, 2, 1]$.

We consider a system where interarrivals of the queues are non-stationary and distrubted $Exp(1/\lambda_i(t))$, described as follows

$$\lambda_1 = \begin{cases} 2.4 & t \leq 100 \\ 0.4 & t > 100 \end{cases}$$

$$\lambda_2 = \begin{cases} 0.6 & t \leq 100 \\ 0.8 & t > 100 \end{cases}$$

$$\lambda_3 = 0.8$$

$$\lambda_4 = \begin{cases} 1.6 & t \leq 100 \\ 0.8 & t > 100 \end{cases}$$

$$\lambda_5 = 0.6$$

The service times are distributed $Exp(1/\mu_{ij})$ where

$$\mu_{ij} = \begin{cases} M_{ij} & i = j \\ 0.8 \cdot M_{ij} & i \neq j \end{cases}$$

After being served, the job leaves the system. We consider a horizon length of $N = 3000$ events, which is around $T = 365$ in time units.