Reinforcement Learning for Datacenter Congestion Control

Chen Tessler, Yuval Shpigelman, Gal Dalal, Amit Mandelbaum, Doron Haritan Kazakov, Benjamin Fuhrer, Gal Chechik, and Shie Mannor

ABSTRACT

We approach the task of network congestion control in datacenters using Reinforcement Learning (RL). Successful congestion control algorithms can dramatically improve latency and overall network throughput. Until today, no such learning-based algorithms have shown practical potential in this domain. Evidently, the most popular recent deployments rely on rule-based heuristics that are tested on a predetermined set of benchmarks. Consequently, these heuristics do not generalize well to newly-seen scenarios. Contrarily, we devise an RL-based algorithm with the aim of generalizing to different configurations of real-world datacenter networks. We overcome challenges such as partial-observability, non-stationarity, and multi-objectiveness. We further propose a policy gradient algorithm that leverages the analytical structure of the reward function to approximate its derivative and improve stability. We show that this scheme outperforms alternative popular RL approaches, and generalizes to scenarios that were not seen during training. Our experiments, conducted on a realistic simulator that emulates communication networks' behavior, exhibit improved performance concurrently on the multiple considered metrics compared to the popular algorithms deployed today in real datacenters. Our algorithm is being productized to replace heuristics in some of the largest datacenters in the world.

1. INTRODUCTION

Most RL algorithms were designed under the assumption that the world can be adequately modeled as a Markov Decision Process (MDP). Unfortunately, this is seldom the case in realistic applications. In this work, we study a real-world application of RL to data-center congestion control. This application is highly challenging because of partial observability and complex multi-objective. Moreover, there are multiple decision-makers that have to make concurrent decisions that can affect each other. We show that even though that the problem at hand does not fit the standard model, we can nevertheless devise an RL framework that outperforms state-of-the-art tailored heuristics. Moreover, we show experimentally that the RL framework generalizes well.

We provide a full description of the Congestion Control (CC) problem in Appendix B, but from a bird's eye view, it is enough to think of CC as a multi-agent, multi-objective, partially observed problem where each decision maker receives a goal (target). The target makes it easy to tune the behavior to fit the requirements (i.e., how latency-sensitive the system is). We devise the target so that leads to beneficial behavior in the multiple considered metrics, without having to tune coefficients of multiple reward components. We model the task of datacenter congestion control as a reinforcement learning problem. We observe that standard algorithms (Mnih et al., 2015; Lillicrap et al., 2015; Schulman et al., 2017) are incapable of solving this task. Thus, we introduce a novel on-policy deterministic-policy-gradient scheme that takes advantage of the structure of our target-based reward function. This method enjoys both the stability of deterministic algorithms and the ability to tackle partially observable problems.

To validate our claims, we develop an RL GYM (Brockman et al., 2016) environment, based on a realistic simulator and perform extensive experiments. The simulator is based on OMNeT++ (Varga, 2002) and emulates the behavior of ConnectX-6Dx network interface cards (state of the art hardware deployed in current datacenters). Our experiments show that our method, Programmable CC-RL (PCC-RL), learns a robust policy, in the sense that it is competitive in all the evaluated scenarios; often outperforming the current state-of-the-art methods.

Our contributions are as follows. (i) We formulate the problem of datacenter congestion control as a partially-observable multi-agent multi-objective RL task. (ii) We present the challenges of this realistic formulation and propose a novel on-policy deterministic-policy-gradient method to solve it. (iii) We provide an RL training and evaluation suite for training and testing RL agents within a realistic simulator. Finally, (iv) we ensure our agent satisfies compute and memory constraints such that it can be easily deployed in future datacenter network devices.

2. REINFORCEMENT LEARNING FOR CONGESTION CONTROL

Due to the lack of space, the background is presented in Appendix A and Appendix C. Below we formulate the problem of CC as an RL task and present our solution.

The RL POMDP framework from Appendix C requires the definition of the four elements in $(\mathcal{S}, \mathcal{A}, P, R)$. The agent, a congestion control algorithm, runs from within the network-interface-card (NIC) and controls the rate of the flows passing through that NIC. At each decision point, the agent observes statistics correlated to the specific flow it controls, an observation o of the state s. The agent then acts by determining

a new transmission rate and observes the outcome of this action.

Observations. As the agent can only observe information relevant to the flow it controls, we consider: the flow's transmission rate, RTT measurement, and number of *NACK* packets received. The NACK packets represent events occurring in the network. A NACK packet signals to the source host that packets have been dropped (e.g., due to congestion) and should be re-transmitted.

Actions. The optimal transmission rate depends both on the number of agents simultaneously interacting in the network, and on the network itself (bandwidth limitations and topology). As such, the optimal transmission rate will vary greatly across scenarios. Since it should be quickly adapted across different orders of magnitude, we define the action as a multiplication of the previous rate. I.e., $\text{rate}_{t+1} = \mathbf{a}_t \cdot \text{rate}_t$.

Transitions. The transition $\mathbf{s}_t \rightarrow \mathbf{s}'_t$ depends on the dynamics of the environment and on the frequency at which the agent is polled to provide an action. Here, the agent acts once an RTT packet is received. This is similar to the definition of a monitor interval by Dong et al. (2018), but while they considered fixed time intervals, we consider event-triggered (RTT) intervals.

Reward. As the task is a multi-agent partially observable problem, the reward must be designed such that there exists a single fixed-point equilibrium. Thus, we let

$$r_t = -\left(\textbf{target} - \frac{\text{RTT}_t^i}{\text{base-RTT}^i} \cdot \sqrt{\text{rate}_t^i}\right)^2 ,$$

where **target** is a constant value shared by all flows, base-RTT^i is defined as the RTT of flow i in an empty system, and RTT_t^i and rate_t^i are respectively the RTT and transmission rate of flow i at time t. $\frac{\text{RTT}_t^i}{\text{base-RTT}^i}$ is also called the rtt inflation of agent i at time t. The ideal reward is obtained when $\textbf{target} = \frac{\text{RTT}_t^i}{\text{base-RTT}^i} \cdot \sqrt{\text{rate}_t^i}$. Hence, when the **target** is larger, the ideal operation point is obtained when $\frac{\text{RTT}_t^i}{\text{base-RTT}^i} \cdot \sqrt{\text{rate}_t^i}$ is larger. The transmission rate has a direct correlation to the RTT, hence the two grow together. Such an operation point is less latency sensitive (RTT grows) but enjoys better utilization (higher rate).

Based on Appenzeller et al. (2004), a good approximation of the RTT inflation in a bursty system, where all flows transmit at the ideal rate, behaves like $\sqrt{N}$, where N is the number of flows. As the system at the optimal point is on the verge of congestion, the major latency increase is due to the packets waiting in the congestion point. As such, we can assume that all flows sharing a congested path will observe a similar $\text{rtt-inflation}_t \approx \frac{\text{RTT}_t^i}{\text{base-RTT}^i}$. As Proposition 1 shows, maximizing this reward results in a fair solution.

Proposition 1. *The fixed-point solution for all N flows sharing a congested path is a transmission rate of $\frac{1}{N}$.*

The proof is provided in Appendix H.

In practice we use the following approximation of the gradient (an exact derivation is provided in Appendix E):

$$\nabla_\theta G^{\pi_\theta}(\mathbf{s}) \approx \tag{1}$$
$$\left[\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \left(\textbf{target} - \text{rtt-inflation}_t \cdot \sqrt{\text{rate}_t}\right)\right] \nabla_\theta \pi_\theta(\mathbf{s}) .$$

Using this derivation enables a deterministic on-policy policy-gradient solution, which we found detrimental to solving this complex task.

3. EXPERIMENTS

To show our method generalizes to unseen scenarios, motivating the use in the real world, we split the scenarios to train and test sets. We train the agents only in the many-to-one domain (see below), on the scenarios: $2 \rightarrow 1, 4 \rightarrow 1$, and $8 \rightarrow 1$. Evaluation is performed on many-to-one, all-to-all, and long-short scenarios. We provide additional details in Appendix G.1, with an extensive overview of the training process, including the technical challenges of the asynchronous CC task, in Appendix G.

3.1 Baseline comparison

The results for the many-to-one tests are presented in Table 1. The simulation settings (number of hosts and flows per host) are presented in Appendix G. We observe that while most methods are competitive at a small scale (small amount of flows with few interactions), at large-scale deployments, all methods aside from PCC-RL encounter extensive periods of packet loss. PCC-RL is the only method capable of maintaining high switch utilization, while keeping the latency low and fairness at a reasonable level.

Table 2: **All-to-all** test results. In these tests, none of the algorithms exhibited packet loss. For 4 hosts, we mark both PCC-RL and DC2QCN as best since it is up to the end-user to determine which outcome is preferred.

Alg.	4 hosts			8 hosts		
	SU	FR	QL	SU	FR	QL
PCC-RL	**94**	**77**	**6**	**94**	**97**	**8**
DC2QCN	**90**	**91**	**5**	91	89	6
HPCC	71	18	3	69	60	3
SWIFT	76	100	11	76	98	13

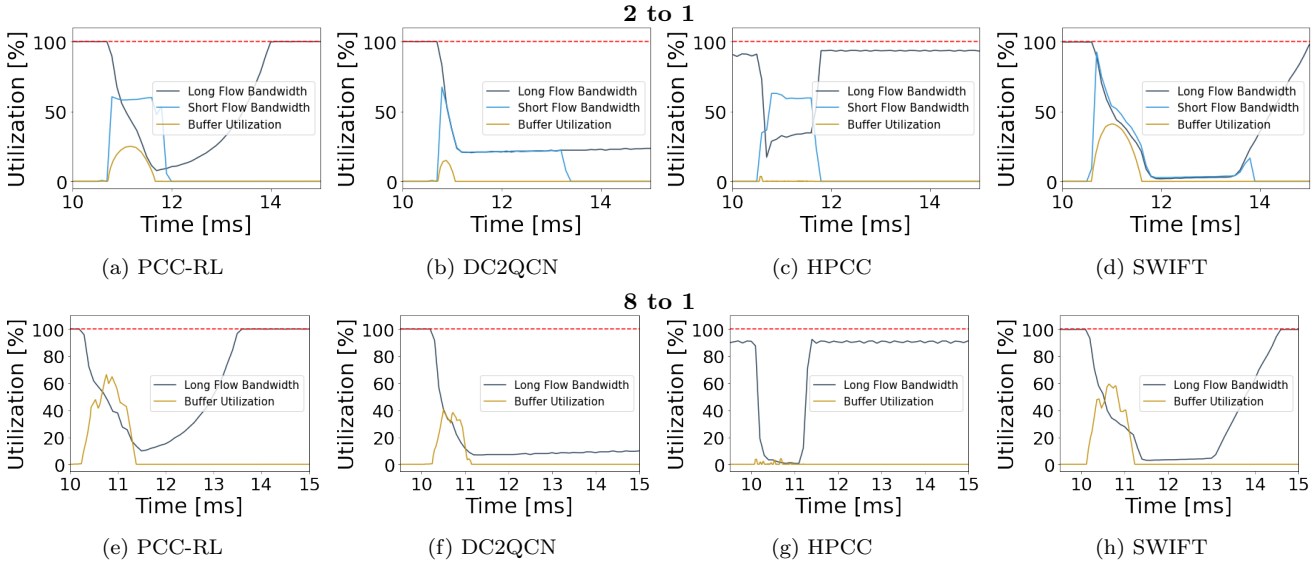
In addition, we evaluate the various methods on an all-to-all setting. Here, there are N hosts and $2 \cdot N$ flows running on each (a total of $2N^2$) . At each host, flow i constantly sends data towards host i mod N through switch i mod N. The results are presented in Table 2. Although SWIFT performed well at low scale many-to-one tests, when transitioning to the all-to-all setting it is incapable of retaining high switch utilization. Similarly to the many-to-one setting, PCC-RL performs competitively and outperforms at the higher scale setting.

Finally, we compared the methods on a long-short setting where the algorithms are tested on their ability to quickly react to changes, presented in Fig. 1 (numerical results in Appendix L). Although PCC-RL did not encounter this scenario during training, it is able to perform competitively. In this scenario, the fastest to react was HPCC, which also maintained minimal buffer utilization. We highlight, though, that HPCC was specifically designed to handle such long-short scenarios (Li et al., 2019). Nonetheless, as opposed to HPCC, PCC-RL achieves 100% utilization before and after the interruption and recovers faster than both SWIFT and DC2QCN.

Summary: We observe that PCC-RL is capable of learning a robust policy. Although in certain tasks the baselines

Table 1: **Many-to-one** test results. Numerical comparison of PCC-RL (our method) with DC2QCN (unpublished followup to Zhu et al. (2015)), HPCC (Li et al., 2019) and SWIFT (Kumar et al., 2020). The column legend is: **SU** Switch Utilization [%] (higher is better), **FR** Fairness defined as $\frac{minrate \cdot 100}{maxrate}$ (higher is better), **QL** Queue Latency [μ sec] (lower is better), and **DR** Drop rate [Gbit/s] (lower is better). As the goal of a CC algorithm is to prevent congestion, we color the tests that failed (extensive periods of packet loss) in red. We mark the best performing methods in each test in bold. For 1024, we mark both DC2QCN and SWIFT as it is up to the end user to determine which outcome is preferred.

Alg.	**128 to 1**				**1024 to 1**				**4096 to 1**				**8192 to 1**			
	SU	FR	QL	DR	SU	FR	QL	DR	SU	FR	QL	DR	SU	FR	QL	DR
PCC-RL	**92**	**95**	**8**	**0**	90	70	15	0	**91**	**44**	**26**	**0**	**92**	**29**	**42**	**0**
DC2QCN	96	84	8	0	**88**	**82**	**17**	**0**	85	67	110	0.2	100	72	157	1.3
HPCC	83	96	5	0	59	48	27	0	73	13	79	0.2	86	8	125	0.9
SWIFT	98	99	40	0	**91**	**98**	**66**	**0**	90	56	120	0.1	92	50	123	0.2



(a) PCC-RL (b) DC2QCN (c) HPCC (d) SWIFT

(e) PCC-RL (f) DC2QCN (g) HPCC (h) SWIFT

Figure 1: **Long Short** test results. The goal is to recover fast, but also avoid packet loss. Higher buffer utilization means higher latency and only when the buffer is fully utilized (100% utilization of the 5MB allocated) packets are dropped. In these tests, none of the algorithms encountered packet loss. The top row (Figs. 1a to 1d) presents the results of a long-short test with 2 flows, and the bottom row (Figs. 1e to 1h) presents a test with 8 flows. We plot the bandwidth utilization of the long flow and the buffer utilization in the switch. Recovery time is measured as the time it takes the long flow to return to maximal utilization. As can be seen, in both scenarios, DC2QCN does not recover within a reasonable time. In addition, in HPCC, the long flow does not reach 100% utilization, even when there are no additional flows.

seldom marginally outperformed it, PCC-RL always obtained competitive performance. In addition, in large scale scenarios, where thousands of flows interact in parallel, we observed that PCC-RL is the only method capable of avoiding packet loss and thus control network congestion. As PCC-RL was trained only on a low-scale scenario, it highlights the ability of our method to generalize and learn a robust behavior, that we believe will perform well in a real datacenter.

4. SUMMARY

In this work, we demonstrated the efficacy and generalization capabilities of RL, in contrast to the hand-crafted algorithms that currently dominate the field of CC. Our experiments utilized the realistic OMNeT++ network simulator that is commonly used to benchmark CC algorithms for deployment in real datacenters. While the various baselines exhibited outstanding performance in certain scenarios, there are others in which they catastrophically failed. Contrarily, PCC-RL learned a robust policy that performed well in all scenarios and often obtained the best results. In addition, we show that PCC-RL generalizes to unseen domains and is the only algorithm capable of operating at a large-scale without incurring packet loss.

PCC-RL was developed with the aim of easy deployment in real datacenters, and potentially even on-device training. We took into consideration memory and compute limitations. By limiting to relatively small and simple networks, combined with int8 quantization, we ensured that the method can run in real-time on a NIC. The next steps involve full productization of the algorithm and adaptive deployment in datacenters – such that enables customization to customers' needs via the tunable target parameter and additional possible reward components.

References

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data

center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, pp. 63–74, 2010.

Appenzeller, G., Keslassy, I., and McKeown, N. Sizing router buffers. *ACM SIGCOMM Computer Communication Review*, 34(4):281–292, 2004.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Dong, M., Meng, T., Zarchy, D., Arslan, E., Gilad, Y., Godfrey, B., and Schapira, M. {PCC} vivace: Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 343–356, 2018.

Jay, N., Rotman, N., Godfrey, B., Schapira, M., and Tamar, A. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pp. 3050–3059, 2019.

Kumar, G., Dukkipati, N., Jang, K., Wassel, H. M., Wu, X., Montazeri, B., Wang, Y., Springborn, K., Alfeld, C., Ryan, M., et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 514–528, 2020.

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

Li, Y., Miao, R., Liu, H. H., Zhuang, Y., Feng, F., Tang, L., Cao, Z., Zhang, M., Kelly, F., Alizadeh, M., et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 44–58. 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, C., Xu, X., and Hu, D. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2014.

Mania, H., Guy, A., and Recht, B. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Mannor, S. and Shimkin, N. A geometric approach to multi-criterion reinforcement learning. *Journal of machine learning research*, 5(Apr):325–360, 2004.

Mittal, R., Lam, V. T., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., and Zats, D. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

Ramakrishnan, K., Floyd, S., and Black, D. Rfc3168: The addition of explicit congestion notification (ecn) to ip, 2001.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *ICML*, 2014.

Spaan, M. T. Partially observable markov decision processes. In *Reinforcement Learning*, pp. 387–414. Springer, 2012.

Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Varga, A. Omnet++ http://www. omnetpp. org. *IEEE Network Interactive*, 16(4), 2002.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M. H., and Zhang, M. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45 (4):523–536, 2015.

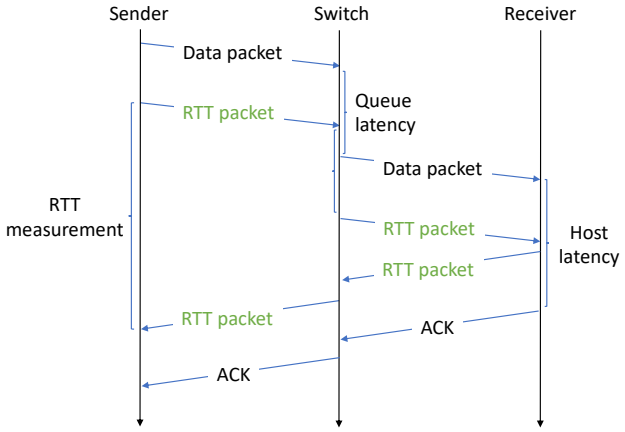


Figure 2: Schema of the packet flow. The sender transmits bursts of data packets and at the end transmits an RTT packet.

APPENDIX

A. NETWORKING PRELIMINARIES

We start with a short introduction covering the most relevant concepts in networking.

In this work, we focus on datacenter networks. In datacenters, traffic contains multiple concurrent data streams transmitting at high rates. The servers, also known as *hosts*, are interconnected through a topology of *switches*. A directional connection between two hosts that continuously transmits data is called a *flow*. We assume, for simplicity, that the path of each flow is fixed.

Each host can hold multiple flows whose transmission rates are determined by a *scheduler*. The scheduler iterates in a cyclic manner between the flows, also known as round-robin scheduling. Once scheduled, the flow transmits a burst of data. The burst's size generally depends on the requested transmission rate, the time it was last scheduled, and the maximal burst size limitation.

A flow's transmission is characterized by two primary values. *Bandwidth*: the average amount of data transmitted, measured in Gbit per second; and *latency*: the time it takes for a packet to reach its destination. *Round-trip-time (RTT)* measures the latency of source→destination→source. While the latency is often the metric of interest, most systems are only capable of measuring RTT. This is presented in Fig. 2.

B. CONGESTION CONTROL

Congestion occurs when multiple flows cross paths, transmitting data through a single congestion point (switch or receiving server) at a rate faster than the congestion point can process. In this work, we assume that all connections have equal transmission rates, as typically occurs in most datacenters. Thus, a single flow can saturate an entire path by transmitting at the maximal rate.

Each congestion point in the network has an inbound buffer enabling it to cope with short periods where the inbound rate is higher than it can process. As this buffer begins to fill, the time (latency) it takes for each packet to reach its destination increases. When the buffer is full, any additional arriving packets are dropped.

B.1 Congestion Indicators

There are various methods to measure or estimate the congestion within the network. The ECN protocol (Ramakrishnan et al., 2001) considers marking packets with an increasing probability as the buffer fills up. Network telemetry is an additional, advanced, congestion signal. As opposed to statistical information (ECN), a telemetry signal is a precise measurement provided directly from the switch, such as the switch's buffer and port utilization.

However, while the ECN and telemetry signals provide useful information, they require specialized hardware. An ideal solution is one that can be easily deployed within existing networks. Such solutions are based on RTT measurements. They measure congestion by comparing the RTT to that of an empty system.

B.2 Objective

CC can be seen as a multi-agent problem. Assuming there are N flows, this results in N CC algorithms (agents) operating simultaneously. Assuming all agents have an infinite amount of traffic to transmit, their goal is to optimize the following metrics:

1. Switch bandwidth utilization – the % from maximal transmission rate.

2. Packet latency – the amount of time it takes for a packet to travel from the source to its destination.

3. Packet-loss – the amount of data (% of maximum transmission rate) dropped due to congestion.

4. Fairness – a measure of similarity in the transmission rate between flows sharing a congested path. We consider $\frac{\min_{\text{flows}} BW}{\max_{\text{flows}} BW} \in [0, 1]$.

The multi-objective problem of the CC agent is to maximize the *bandwidth utilization* and *fairness*, and minimize the *latency* and *packet-loss*. Thus, it may have a Pareto-front (Liu et al., 2014) for which optimality w.r.t. to one objective may result in sub-optimality of another. However, while the metrics of interest are clear, the agent does not necessarily have access to signals representing them. For instance, fairness is a metric that involves all flows, yet the agent observes signals relevant only to the flow it controls. Hence, it is impossible for a flow to obtain an estimate of the current fairness in the system. Instead, we reach fairness by setting each flow's individual target adaptively, based on known relations between its current RTT and rate. More details on this are given in Sec. 2.

This problem exhibits additional complexities. As the agent only observes information relevant to the flow it controls, this task is partially observable. The observation might lack sufficient statistics required to determine the optimal policy. Moreover, the network's reaction to transmission rate changes is delayed by $\mathcal{O}(RTT)$.

C. REINFORCEMENT LEARNING PRELIMINARIES

We model the task of congestion control as a multi-agent partially-observable multi-objective MDP, where all agents share the same policy. Each agent observes statistics relevant to itself and does not observe the entire global state (e.g., the number of active flows in the network).

We consider an infinite-horizon Partially Observable Markov Decision Process (POMDP). A POMDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, P, R)$ (Puterman, 1994; Spaan, 2012). An agent interacting with the environment observes a state $\mathbf{s} \in \mathcal{S}$ and performs an action $\mathbf{a} \in \mathcal{A}$. After performing an action, the environment transitions to a new state $\mathbf{s}'$ based on the transition kernel $P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ and receives a reward $r(\mathbf{s}, \mathbf{a}) \in R$. In a POMDP, the observed state does not necessarily contain sufficient statistics for determining the optimal action.

We consider the average reward metric, defined as follows. We denote Π as the set of stationary deterministic policies on $\mathcal{A}$, i.e., if $\pi \in \Pi$ then $\pi : \mathcal{S} \to \mathcal{A}$. Let $\rho^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the gain of a policy π, defined in state $\mathbf{s}$ as $\rho^\pi(\mathbf{s}) \equiv \lim_{T\to\infty} \frac{1}{T}\mathbb{E}^\pi[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \mid \mathbf{s}_0 = \mathbf{s}]$, where $\mathbb{E}^\pi$ denotes the expectation w.r.t. the distribution induced by π.

The goal is to find a policy π^*, yielding the optimal gain ρ^*, i.e., for all $\mathbf{s} \in \mathcal{S}$, $\pi^*(\mathbf{s}) \in \arg\max_{\pi\in\Pi} \rho^\pi(\mathbf{s})$ and the optimal gain is $\rho^*(\mathbf{s}) = \rho^{\pi^*}(\mathbf{s})$. A well known result Puterman (1994)[Theorem 6.2.10], is that there always exists an optimal policy which is stationary and deterministic.

D. RELATED WORK

Hand-tuned CC: While there has been a vast amount of work on congestion control, we focus on datacenter congestion control methods. Previous work tackled this problem from various angles. Alizadeh et al. (2010) used a TCP-like protocol, which increases the transmission until congestion is sensed, and then dramatically decreases it. Mittal et al. (2015); Kumar et al. (2020) directly used RTT to react quickly to changes. Zhu et al. (2015) utilized the ECN protocol, a statistical signal provided by the switch, and Li et al. (2019) added telemetry information that requires specialized hardware yet proves to benefit greatly in terms of reaction times.

Optimization-based CC: Although most previous work has focused on hand-tuned algorithmic behavior, two notable mentions have taken an optimization-based approach. Dong et al. (2018) presented the PCC-Vivace algorithm, which combines information from fixed time intervals such as bandwidth, latency inflation, and more. As it tackles the problem via online convex optimization, it is stateless; as such, it does not optimize for long-term behavior but rather focuses on the immediate reward (bandit setting). This work was then extended in the Aurora system (Jay et al., 2019). Aurora provides the monitor interval, defined in PCC-Vivace, as a state for a PPO (Schulman et al., 2017) algorithm. Although these methods work in a naive single-agent setting, we observed their inability to converge to satisfying behavior in the realistic multi-agent setting.

Multi-objective RL: This task can also be cast as a multi-objective RL problem (Mannor & Shimkin, 2004) and solved by combining the various metrics into a scalar reward function (Jay et al., 2019) (e.g., $r = a \cdot \text{bandwidth} - b \cdot \text{latency} - c \cdot \text{packet loss}$, where a, b, c are positive constant coefficients). However, in practice, the exact coefficients are selected through a computationally-intensive process of hyper-parameter tuning. Previous work (Leike et al., 2017; Mania et al., 2018; Tessler et al., 2018) has shown that these coefficients do not generalize: a coefficient that leads to a satisfying behavior on one domain may lead to catastrophic failure on the other. We propose an alternative approach, in which we present a reward function that is domain agnostic. Our reward has a single fixed point solution, which is optimal for any domain.

E. IMPLEMENTATION

Thriving for simplicity, we initially attempted to solve this task using standard RL techniques such as DQN (Mnih et al., 2015), DDPG (Lillicrap et al., 2015) and PPO (Schulman et al., 2017). Due to the challenges this task exhibits, namely partial observability and multi-agent optimization, these methods did not converge. Additionally, we experimented with Aurora (Jay et al., 2019): an RL agent for CC, designed to solve a single-agent setup. As such, it did not converge in our more complex domain. All the above attempts are documented in detail in Appendix I.

Due to the partial observability, on-policy methods are the most suitable. And as the goal is to converge to a stable multi-agent equilibrium, and due to the high-sensitivity action choice, deterministic policies are easier to manage. Thus, we

devise a novel on-policy deterministic policy-gradient (Silver et al., 2014, DPG) method that directly relies on the structure of the reward function as given below. In DPG, the goal is to estimate $\nabla_\theta G^{\pi_\theta}$, the gradient of the value of the current policy, with respect to the policy's parameters θ. By taking a gradient step in this direction, the policy is improving and thus under standard assumptions will converge to the optimal policy.

As opposed to off-policy methods, on-policy learning does not demand a critic. We observed that due to the challenges in this task, learning a critic is not an easy feat. Hence, we focus on estimating $\nabla_\theta G^{\pi_\theta}$ from a sampled trajectory.

$$\begin{aligned}\nabla_\theta G^{\pi_\theta} =& \nabla_\theta \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \pi_\theta(\mathbf{s}_t))\right] \\ &= \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \nabla_{\mathbf{a}} r(\mathbf{s}_t, \mathbf{a})|_{\mathbf{a}=\mathbf{a}_t} \cdot \nabla_\theta \pi_\theta(\mathbf{s}_t) \\ &= -\lim_{T\to\infty} \frac{1}{T} \\ &\quad \cdot \sum_{t=0}^{T} \nabla_{\mathbf{a}} \left(\mathbf{target} - \text{rtt-inflation}^i \cdot \sqrt{\text{rate}_t^i}\right)^2 |_{\mathbf{a}=\mathbf{a}_t} \\ &\quad \cdot \nabla_\theta \pi_\theta(\mathbf{s}_t)\,.\end{aligned} \tag{2}$$

Using the chain rule we can estimate the gradient of the reward $\nabla_{\mathbf{a}} r(\mathbf{s}_t, \mathbf{a})$:

$$\begin{aligned}\nabla_{\mathbf{a}} r(\mathbf{s}_t, \mathbf{a}) = &\left(\mathbf{target} - \text{rtt-inflation}_t(\mathbf{a}) \cdot \sqrt{\text{rate}_t(\mathbf{a})}\right) \\ &\cdot \nabla_{\mathbf{a}} \left(\text{rtt-inflation}_t(\mathbf{a}) \cdot \sqrt{\text{rate}_t}(\mathbf{a})\right).\end{aligned} \tag{3}$$

Notice that both $\text{rtt-inflation}_t(\mathbf{a})$ and $\sqrt{\text{rate}_t(\mathbf{a})}$ are monotonically increasing in $\mathbf{a}$. The action is a scalar determining by how much to change the transmission rate. A faster transmission rate also leads to higher RTT inflation. Thus, the signs of $\text{rtt-inflation}_t(\mathbf{a})$ and $\sqrt{\text{rate}_t(\mathbf{a})}$ are identical and $\nabla_{\mathbf{a}} \left(\text{rtt-inflation}_t(\mathbf{a}) \cdot \sqrt{\text{rate}_t(\mathbf{a})}\right)$ is always non-negative.

However, estimating the exact value $\nabla_{\mathbf{a}}(\text{rtt-inflation}_t(\mathbf{a}) \cdot \sqrt{\text{rate}_t(\mathbf{a})})$ is impossible given the complex dynamics of a datacenter network. Instead, as the sign is always non-negative, we approximate this gradient with a positive constant which can be absorbed into the learning rate.

$$\begin{aligned}&\nabla_\theta G^{\pi_\theta}(\mathbf{s}) \approx \\ &\left[\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \left(\mathbf{target} - \text{rtt-inflation}_t \cdot \sqrt{\text{rate}_t}\right)\right] \nabla_\theta \pi_\theta(\mathbf{s})\,.\end{aligned} \tag{4}$$

In layman's terms – if $\text{rtt-inflation}_t * \sqrt{\text{rate}_t}$ is above the target, the gradient will push the action towards decreasing the transmission rate, and vice versa. As all flows observe approximately the same rtt-inflation_t, the objective drives them towards the fixed-point solution. As shown in Proposition 1, this occurs when all flows transmit at the same rate of $\frac{1}{N}$ and the system is slightly congested as proposed by Kumar et al. (2020).

Finally, the true estimation of the gradient is obtained for $T \to \infty$. Our approximation for this gradient is by averaging over a finite, sufficiently long, T. In practice, T is determined empirically.

F. THE CHALLENGES OF REAL-WORLD DEPLOYMENT

PCC-RL currently undergoes a productization procedure in a large tech company (>15K employees) and is to be deployed in its live datacenters. Thus, beyond training the RL agent, our goal is to provide a method that can run in real-time on-device (on a NIC). This requirement presents two limitations. (1) The problem is asynchronous. While the CC algorithm is determining the transmission rate limitation, the flows continue to transmit data. As such, decision making must be efficient and fast such that inference can be performed within $\mathcal{O}(RTT) = \mathcal{O}(\mu sec)$. (2) Each NIC can control thousands of flows. As we require real-time reaction, the algorithm must utilize fast, yet small, memory. Hence, the amount of memory stored per each flow must be minimal.

Thus, for the policy we choose a neural network that is composed of 2 fully connected layers ($in \to 32 \to 16$) followed by an LSTM ($16 \to 16$) and a final fully connected layer ($16 \to 1$). As the information takes $\mathcal{O}$(RTT) to propagate through the network, the next state is also a function of the previous actions, we observed that the LSTM was crucial. Such an architecture, combined with ReLU activation functions, enables fast inference using common deep learning accelerators (DLA).

While a small number of weights and a low memory footprint enables faster inference, to meet the strict requirements of running in real-time on-device, we also quantize a trained agent and analyze its performance. This is discussed in detail in **??**.

G. SIMULATOR

The simulator attempts to model the network behavior as realistically as possible. The task of CC is a multi-agent problem, there are multiple flows running on each host (server) and each flow is unaware of the others. As such, each flow is a single agent, and 4096 flows imply 4096 concurrent agents.

Each agent is called by the CC algorithm to provide an action. The action, whether continuous or discrete, is mapped to a requested transmission rate. When the flow is rescheduled, it will attempt to transmit at the selected rate. Calling the agent (the triggering event) occurs each time an RTT packet arrives.

Agents are triggered by spontaneous events rather than at fixed time intervals; this makes the simulator asynchronous. Technically speaking, as the simulator exposes a single step function, certain agents might be called upon more times than others.

While the action sent to the simulator is for the current state s_t, in contradiction to the standard GYM environments, state s_{t+1} is not necessarily from the same flow as s_t. Due to the asyncronous nature of the problem, the simulator provides the state which corresponds to the next agent that receives an RTT packet.

To overcome this, we propose a 'KeySeparatedTemporalReplay', a replay memory that enables storing asynchronous rollouts separated by a key (flow). We utilize this memory for training our method and Aurora (PPO), who both require gradient calculation over entire rollouts.

G.1 Experimental Details

We compare to 3 baseline algorithms: DC2QCN that utilizes the ECN packet marking protocol, HPCC that focuses on network telemetry information, and SWIFT that can be deployed in any existing datacenter, as it relies only on RTT measurements. As these methods lack an official implementation, we use unofficial implementations. Although unofficial, these implementations are currently deployed and used daily in real datacenters[1].

Many-to-one: Denoted by $N \to 1$, this scenario emulates N senders transmitting data through a single switch to a single receiver. We evaluate the agents on $2^i \to 1$, for $i \in \{4, 5, \ldots, 13\}$. The exact configuration (number of flows per host) is presented in Appendix G.

All-to-all: This scenario emulates multiple servers transmitting data to all other servers. In this case, given there are N servers, there will also be N congestion points. All data sent towards server i routes through switch port i. This synchronized traffic causes a high system load. While the 'many-to-one' is a relatively clean setting, 'all-to-all' tests the ability of the various algorithms to cope in a complex and dynamic system.

Long-short: In addition to testing the ability of the algorithms to converge to a stable-point solution, the long-short scenario evaluates the ability of each agent to dynamically react to changes. A single flow (the 'long' flow) transmits an infinite amount of data, while several short flows randomly interrupt it with short data transmission. The goal is to test how fast the long flow reacts and reduces its transmission rate to enable the short flows to transmit their data. Once the short flows are finished, the long flow should recover quickly to full line rate. We follow the process from interruption until full recovery. We present an example of ideal behaviors in Fig. 3.

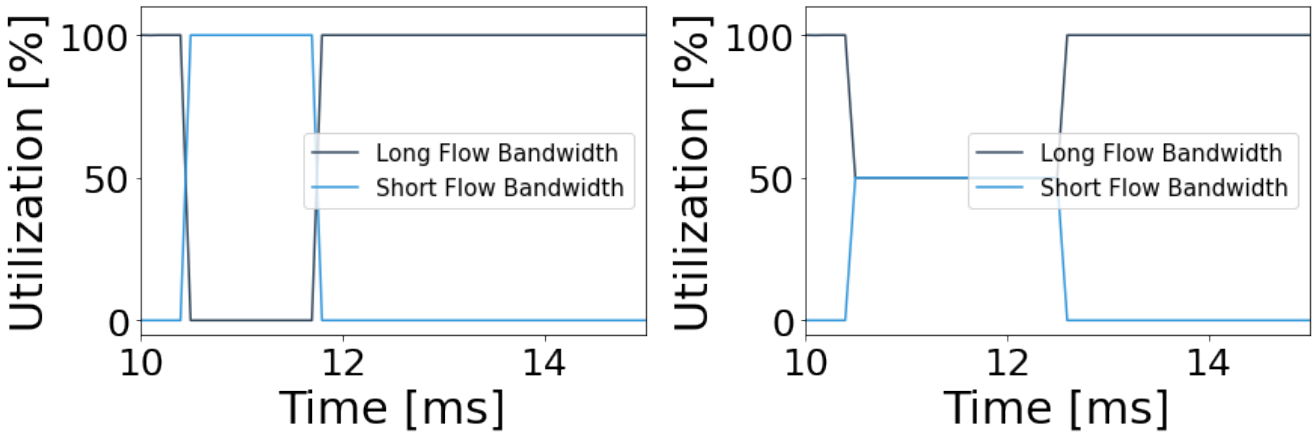


Figure 3: **Long Short *ideal behavior***. The above plots depict two ideal behaviors. On the left, the long flow halts while the short flow is transmitting data. On the right, the long and short flows quickly converge to an equal fair transmission rate. Both solutions are ideal, and determining which solution is preferable depends on the datacenter requirements.

G.2 Many to one tests

In the many to one tests, each configuration combines a different number of hosts and flows. For completeness we provide the exact mapping below:

[1]A reference shall appear in the final version and is omitted here to guarantee anonymity.

Table 3: Many to one experiment mapping

Total flows	Hosts	Flows per per host
2	2	1
4	4	1
16	16	1
32	32	1
64	64	1
128	64	2
256	32	8
512	64	8
1024	32	32
2048	64	32
4096	64	64
8192	64	128

G.3 Computational Details

The agents were trained on a standard i7 CPU with 6 cores and a single GTX 2080. The training time (for 200k steps) took 2-3 hours. The major bottleneck was the evaluation times. We evaluated the agents in a many-to-one setting with a very high number of flows (up to 8k). The more flows, the longer the test. In the python version, on this system it took approximately 2 days to evaluate the agent throughout 2 simulated seconds. However, on an optimized C implementation, the same evaluation took 20 minutes.

H. FIXED-POINT PROOF

Proposition 2. *The fixed-point solution for all N flows sharing a congested path is a transmission rate of* $\frac{1}{N}$.

The proof relies on the assumption that all flows sharing a congested path observe the same rtt inflation. Although the network topology affects the packet path and thus the latency, this latency is minimal when compared to the queue latency of a congested switch.

Proof.
The maximal reward is obtained when all agents minimize the distance $||\text{rtt-inflation} \cdot \sqrt{rate} - \textbf{target}||$. There are two stationary solutions (1) $\text{rtt-inflation} \cdot \sqrt{rate} < \textbf{target}$ or (2) $\text{rtt-inflation} \cdot \sqrt{rate} = \textbf{target}$.

As flows can always reduce the transmission rate (up to 0) and $\text{rtt-inflation} \propto \sqrt{rate}$. A solution where $\text{rtt-inflation} \cdot \sqrt{rate} > \textbf{target}$ is not stable.

We analyze both scenarios below and show that a stable solution at that point is also fair.

1. $\text{rtt-inflation} \cdot \sqrt{rate} < \textbf{target}$. The value is below the target. Minimizing the distance to the target means maximizing the transmission rate. A stable solution below the target is obtained when the flows are transmitting at full-line rate (can't increase the rate over 100%) and yet the rtt-inflation is low (small or no congestion). As all flows are transmiting at 100% this solution is fair.

2. $\text{rtt-inflation} \cdot \sqrt{rate} = \textbf{target}$. For any i, j sharing a congested path, we assume that $\text{rtt-inflation}_i = \text{rtt-inflation}_j$, this is a reasonable assumption in congested systems as the RTT is mainly affected by the latency in the congestion point. As all flows observe $\text{rtt-inflation} \cdot \sqrt{rate} = \textbf{target}$, we conclude that if $\text{rtt-inflation} \cdot \sqrt{rate} = \textbf{target}$ then $\sqrt{rate_i} = \sqrt{rate_j} = \frac{1}{N}, \forall i, j$.

I. TRAINING CURVES

In this section of the appendix, we expand on additional methods that did not prevail as well as our algorithm.

I.1 Aurora

We begin with Aurora (Jay et al., 2019). Aurora is similar to PCC-RL in how it extracts statistics from the network. A monitor-interval (MI) is defined as a period of time over which the network collects statistics. These statistics, such as average drop rate, RTT inflation, and more, are combined into the state provided to the agent.

However, Aurora focused on the task of single-agent congestion control. As they considered internet congestion control (as opposed to datacenter congestion control), their main challenge was handling jitters (random noise in the network resulting in packet loss even at network under-utilization).

An additional difference is that Aurora defines a naive reward signal, inspired by PCC-Vivace (Dong et al., 2018):

$$r = a \cdot BW - b \cdot RTT - c \cdot DROPRATE \ \ , a, b, c \geq 0$$

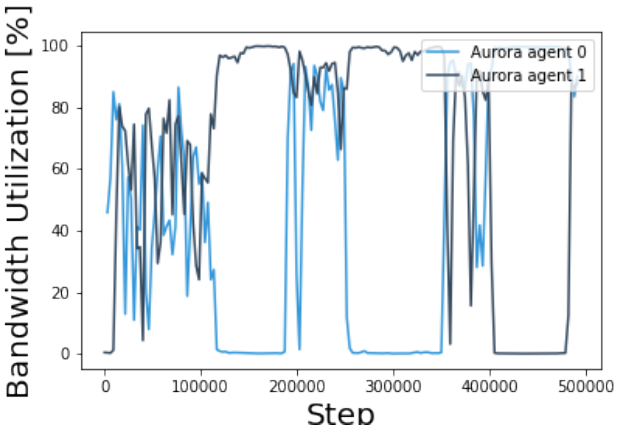


Figure 4: **Aurora training:** two hosts with a single flow per host. The flows are unable to converge to a stable equilibrium.

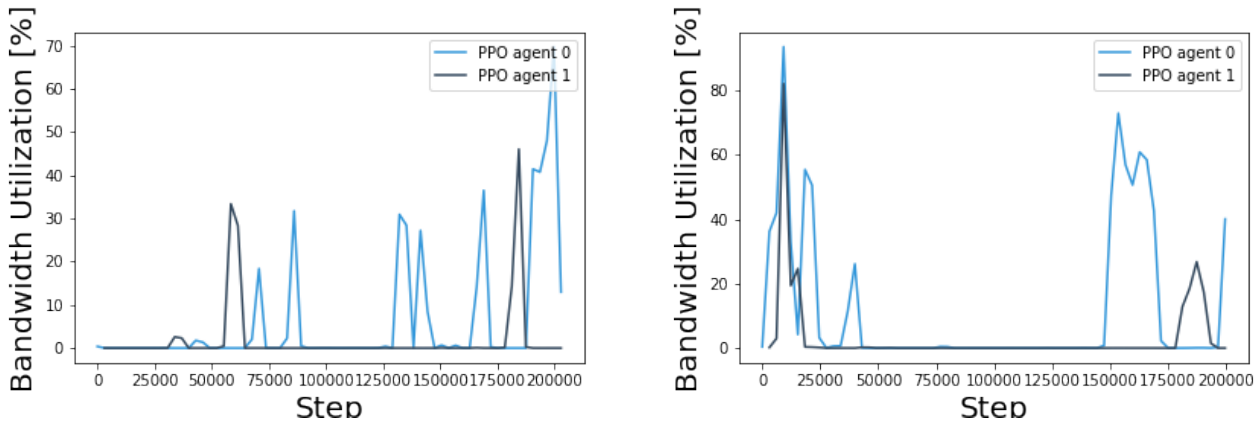
We observe in Fig. 4 that Aurora is incapable of converging in the simulated environment. We believe this is due to the many challenges the real world exhibits, specifically partial observability and the non-stationarity of the opposing agents.

I.2 PPO

PCC-RL introduces a deterministic on-policy policy-gradient scheme that utilizes specific properties of the reward function. It is not immediately clear why such a scheme is important. As such, we compare to PPO trained on our raw reward function

$$r = -(\textbf{target} - \text{rtt-inflation} \cdot \sqrt{\text{rate}})^2$$

We present two versions of PPO. (1) A continuous action space represented as a stochastic Gaussian policy $a \sim \mathcal{N}(\mu, \sigma)$, $\mu \in [0.8, 1.2]$ (as is common in continuous control tasks such as MuJoCo (Todorov et al., 2012; Schulman et al., 2017)). (2) A discrete action space represented as a stochastic discrete policy (softmax) where $a \in \{0.8, 0.95, 1, 1.05, 1.1, 1.2\}$.



(a) Continuous

(b) Discrete

Figure 5: **PPO training:** both the continuous (Fig. 5a) and discrete (Fig. 5b) versions of the PPO algorithm are unable to learn, even with our target-based reward signal.

I.3 PCC-RL

Finally, we present the training curves of PCC-RL. As can be seen, Fig. 6, PCC-RL quickly converges to a region of the fixed-point stable equilibrium.

Table 4: **Many-to-one:** Numerical comparison of PCC-RL with various selected operation points, where Strict, Standard, and Loose refer to targets of 1, 2, and 20, respectively. The 'Standard' results are those presented in the previous section. Here, there isn't a single 'best' solution. The prefered solution depends on the datacenter requirements.

Target	8192 to 1			
	SU	FR	QL	DR
Strict	**51**	**99**	**12**	**0**
Standard	**92**	**29**	**42**	**0**
Loose	**92**	**35**	**55**	**0**

Table 5: **All-to-all:** Numerical comparison of PCC-RL with various operation points. The standard results are presented in the previous section.

Target	4 hosts			8 hosts		
	SU	FR	QL	SU	FR	QL
Strict	60	30	4	74	52	13
Standard	**94**	**77**	**6**	**94**	**97**	**8**
Loose	73	18	8	71	21	9

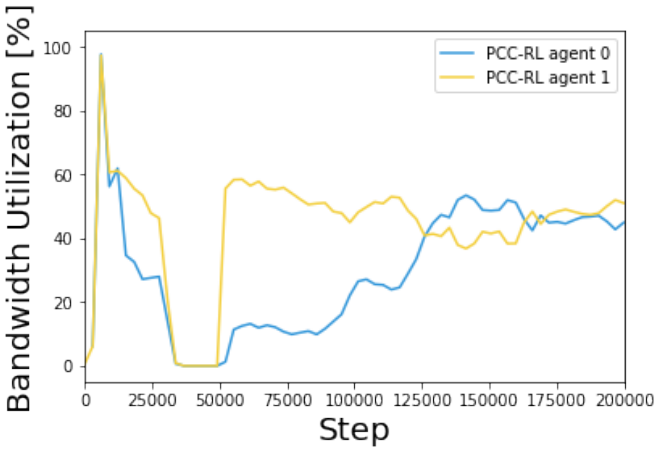


Figure 6: **PCC-RL training:** the agents quickly converge to a region of the fair equilibrium.

J. SELECTING THE OPERATION POINT

As we have seen in Tables 1 and 2 and Fig. 1, the various algorithms exhibit different behaviors. For instance, as seen in Table 1 under the *1024 to 1* evaluation, both DC2QCN and SWIFT obtain good results. It is not clear whether one is absolutely better than the other. The decision depends on the use case. Certain datacenters are very sensitive to latency and would thus prefer DC2QCN. Others might prefer behavior such as of SWIFT, that provides higher fairness and bandwidth outputs.

PCC-RL is controlled by a single parameter, **target**. By changing the target, the CC behavior in the datacenter is adapted. When the required behavior is low latency, the target value should be set close to 0, whereas in a system is less latency sensitive, while improved fairness and utilization can be achieved by setting higher values.

The results in Tables 4 to 6 present an interesting image on how PCC-RL behaves when the operation point is changed. We compare three operation points: 'strict', 'standard', and 'loose', corresponding to targets 1, 2, and 20, respectively.

As expected, the strict target results in lower latency while the opposite occurs for the loose. As the agent attempts to maintain a stricter latency profile, it is required to keep the switch's buffer utilization as low as possible. This results in a dramatic decrease in switch utilization; see Table 6. On the other hand, a looser profile suffers from higher latency but is capable of reacting faster to the appearance of new flows, which can be explained by the new flows joining faster due to the larger "allowance", and attains better fairness across flows (Table 4).

K. QUANTIZATION

A major challenge in CC is the requirement for real-time decisions. Although the network itself is relatively small and efficient, when all computations are performed in int8 data type, the run-time can be dramatically optimized.

To this end, we test the performance of a trained PCC-RL agent after quantization and optimization in C. Following the methods in Wu et al. (2020), we quantize the network to int8. Combining int8 operations requires a short transition to int32

Table 6: **Long-short:** Numerical comparison of PCC-RL with various operation points. The 'Loose' results are those presented in the previous section. When scaling to above 1024 flows, both the strict and standard settings are incapable of recovering in a reasonable amount of time.

Target \ Flows	2	128	1024	2048
Strict	5e-2	5e-2	-	-
Standard	5e-3	5e-3	-	-
Loose	**6e-7**	**8e-4**	**3e-2**	**3e-2**

(to avoid overflow), followed by a de-quantization and re-quantization step.

We present the results in Table 7. The quantized agent performs similarly to an agent trained on a *loose* target (improved switch utilization at the expense of a slightly higher latency). These exciting results show that a quantized agent is capable of obtaining similar performance to the original agent. This is a major step forward towards deployment in live datacenters.

Table 7: **Quantization many to one:** Comparison of the original Python PCC-RL agent with an optimized and quantized C version.

Flows	Switch Utilization		Fairness		Queue Latency	
	Original	Quantized	Original	Quantized	Original	Quantized
2	96.9	99.91	0.99	1.00	5.2	8.29
4	96.9	99.86	0.98	1.00	5.4	8.59
16	95.1	99.61	0.99	0.99	6.2	9.46
32	95.6	99.22	0.86	0.99	7.0	9.90
64	93.3	99.06	0.96	0.98	7.0	9.72
128	92.5	98.57	0.94	0.96	8.0	10.94
256	91.4	98.02	0.91	0.90	9.3	12.85
512	90.4	97.54	0.86	0.86	11.3	16.26
1024	90.2	97.10	0.74	0.75	14.7	20.92
2048	90.5	96.74	0.56	0.61	20.3	27.67
4096	91.3	96.65	0.46	0.43	27.7	36.87
8192	92.8	96.79	0.28	0.29	40.0	48.40

L. LONG SHORT DETAILS

Table 8: **Long-short:** The results represent the time it takes from the moment the *short* flows interrupt and start transmitting until they finish and the *long* flow recovers to full line rate transmission. DC2QCN does not recover fast enough and has thus failed the high scale recovery tests. We present the recovery time **RT** (μsec), drop rate **DR** (Gbit/s) and the bandwidth utilization of the long flow **LBW** (%).

Algorithm	2 flows			128 flows			1024 flows			2048 flows		
	RT	DR	LBW	RT	DR	LBW	RT	DR	LBW	RT	DR	LBW
PCC-RL	**6e-7**	**0**	**97**	**8e-4**	**0**	**94**	3e-2	1.1	62	3e-2	2.7	56
DC2QCN	3e-2	0	63	5e-2	0	40	-			-		
HPCC	3e-5	0	90	1e-2	0.4	75	2e-2	1.1	72	3e-2	1.8	62
SWIFT	1e-3	0	97	1e-2	0.3	85	**1e-2**	**1.2**	**83**	**2e-2**	**2.1**	**72**