Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control

Fabian Ruffy[∗ †]
fruffy@cs.ubc.ca

Michael Przystupa[∗ †]
bot267@cs.ubc.ca

Ivan Beschastnikh[†]
bestchai@cs.ubc.ca

Abstract

Recent networking research has identified that data-driven congestion control (CC) can be more efficient than traditional CC in TCP. Deep reinforcement learning (RL), in particular, has the potential to learn optimal network policies. However, RL suffers from instability and over-fitting, deficiencies which so far render it unacceptable for use in datacenter networks. In this paper, we analyze the requirements for RL to succeed in the datacenter context. We present a new emulator, Iroko, which we developed to support different network topologies, congestion control algorithms, and deployment scenarios. Iroko interfaces with the OpenAI gym toolkit, which allows for fast and fair evaluation of different RL and traditional CC algorithms under the same conditions. We present initial benchmarks on three deep RL algorithms compared to TCP New Vegas and DCTCP. Our results show that these algorithms are able to learn a CC policy which exceeds the performance of TCP New Vegas on a dumbbell and fat-tree topology. We make our emulator open-source and publicly available: `https://github.com/dcgym/iroko`.

1 Introduction

Reinforcement learning (RL) has seen a surge of interest in the networking community. Recent contributions include data-driven flow control for wide-area networks [15], job scheduling [30], and cellular congestion control [57]. A particularly promising domain is the data center (DC) as many DC networking challenges can be formulated as RL problems [46]. Researchers have used RL to address a range of DC tasks such as routing [50, 9], power management [49], and traffic optimization [12].

Adhering to the objective of maximizing future rewards [47], RL has the potential to learn anticipatory policies. Data center CC, can benefit from this feature, as current DC flow control protocols and central schedulers are based on the fundamentally reactive TCP algorithm [54, 34]. While many techniques are designed to respond to micro-bursts or flow collisions as quickly as possible, they are not capable of preemptively identifying and avoiding these events [21, 13]. Any time flows collide, packets and goodput is lost. Given the availability of data and the range of RL algorithms, CC is an excellent match for RL.

However, a lack of generalizability [25, 37, 31, 52, 58] and reproducibility [20] makes RL an unacceptable choice for DC operators, who expect stable, scalable, and predictable behavior. Despite these limitations, RL is progressing quickly in fields such as autonomous driving [39] and robotics [24]. These domains exhibit properties similar to DC control problems: both deal with a large input space and require continuous output actions. Decisions have to be made rapidly (on the order of microseconds) without compromising safety and reliability.

What these fields have, and what current DC research is missing, is a common platform to compare and evaluate techniques. RL benchmark toolkits such as the OpenAI gym [10] or RLgarage (formerly

[∗] Equal contribution

[†] University of British Columbia

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

RLlab) [16] foster innovation and enforce a common standards framework. In the networking space, the Pantheon project [56] represents a step in this direction. It provides a system to compare CC solutions for wide area networks. No such framework currently exists for DCs, partially because topology and traffic patterns are often considered private and proprietary [7].

We contribute **Iroko**, a DC emulator to understand the requirements and limitations of applying RL in DC networks. Iroko interfaces with the OpenAI gym [10] and offers a way to fairly evaluate centralized and decentralized RL algorithms against conventional traffic control solutions.

As preliminary evaluation, we compare three existing RL algorithms in a dumbbell and a fat-tree topology. We also discuss the design of our emulator, as well as limitations and challenges of using reinforcement learning in the DC context.

2 The Data Center as RL Environment

A major benefit of datacenters is the flexibility of deployment choices. Since a DC operator has control over all host and hardware elements, they can manage traffic at fine granularity. An automated agent has many deployment options in the data center. Common techniques to mitigate congestion include admission control [34, 13], load-balancing of network traffic [2, 8], queue management [18, 6], or explicit hardware modification [4, 3]. As TCP is inherently a self-regulating, rate-limiting protocol, our emulator uses admission control to moderate excess traffic.

2.1 Patterns of Traffic

In order for an algorithm to operate proactively, it needs to be able to predict future network state. It is unclear how trainable DC traffic truly is, as public data is few and far between. However, prior work indicates that repeating patterns do exist [22, 38, 32, 8, 45].

Although the use of online-learning algorithms for the Internet is controversial [40], PCC [15] and Remy [55] have demonstrated that congestion control algorithms that evolve from trained data can compete with or even exceed conventional, manually tuned algorithms. The idea of drawing from *past* data to learn for the *future* is attractive. It is particularly viable in data centers, which are fully observable, exhibit specific application patterns [22], and operate on recurrent tasks.

If DC traffic is sufficiently predictable, it is possible to design a proactive algorithm which forecasts the traffic matrix in future iterations, and accordingly adjusts host sending rates. We agree with prior debate that a local, greedily optimizing algorithm may not be capable of achieving this goal [40]. Instead, utility needs to be maximized by leveraging global knowledge, either over a centralized solution such as FastPass [34] or distributed in terms of a message passing solution such as PERC [21].

2.2 Decentralized vs Centralized Control

Control techniques are either centralized or decentralized. TCP, for example, is a decentralized control algorithm. Each host optimizes traffic based on its local control policy and tries to maximize its own utility. Recent data-driven variants of TCP include PCC [15] or Copa [5], which actively learn an optimal traffic policy based on local metrics.

A decentralized control scheme offers the advantage of scalability and reliability when handling long flows with extend round trip times (RTTs). DCs in contrast typically exhibit short and bursty flows with low RTTs, which limits convergence [21]. This limitation frequently leads to inefficient flow utilization or instantaneous queue build up [34, 13, 54].

In contrast to the decentralized control of TCP, a centralized policy can use the global information to efficiently manage network nodes. Recent examples include the FastPass [34] arbiter, the Auto [12] traffic manager, or the Hedera [2] flow scheduler. A major concern of centralized systems is signaling latency delay and limitations in processing power. However, a central scheme has the potential to plan ahead and asynchronously grants hosts traffic guarantees based on its current anticipated model of the network. MicroTE [8], which optimizes routing by predicting the next short-term traffic matrix, represents an instantiation of this model.

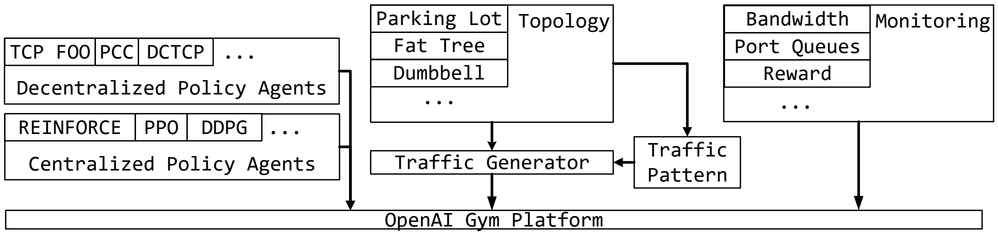


Figure 1: Architecture of the Iroko emulator.

2.3 Sources of Information

The available options for network data acquisition in a DC range from switch statistics, application flows, job deployment monitoring, or even explicit application notifications.

Our algorithms use metrics from the transport layer and below, which have traditionally been used in TCP congestion control algorithms. Modeling an objective function based only on congestion signals is a tried and tested approach. Remy [55] and PCC [15] have demonstrated that it is possible to dynamically learn and improve the congestion function from simple network feedback.

Theoretically, it is possible to query for switch buffer occupancy, packet drops, port utilization, active flows, and RTT. End-hosts can provide metrics in goodput, latency, jitter, and individual loss.

As improvement over the packet- and delay-based TCP, relative increases in RTT have been effectively used as a signal in congestion avoidance research [33, 11]. Queue length in switch interfaces is a discrete value and precedes an increase in RTT, making it an equivalent congestion metric to RTT increase.

Measured throughput represents the current utilization of the network and acts as a metric of the actual utility for an active policy (a network without traffic has zero queuing, after all). A one-hot encoding of active TCP/UDP flows per switch-port can serve as basis to identify network patterns.

3 Emulator Design

We designed Iroko to be extensible and modular. All DC information is abstracted away from the RL agent, providing flexibility in data acquisition and modeling. The testing environment is assembled by combining a set of core components. This includes a network topology, a traffic generator, monitors for collecting data center information, and an agent to enforce the congestion policy (see Figure 1). The emulator's flexibility allows it to support centralized arbiters as well as decentralized, host-level CC approaches. In general, decentralized agents represent traditional TCP algorithms such as DCTCP [4], TIMELY [33], or PCC [15], while centralized agents operate as RL policies.The emulator also supports hybrid deployments, which could operate as multi-agent systems as described in [46]. The monitors feed information to the agent or record data for evaluation. The topology defines the underlying infrastructure and traffic patterns that the DC hosts will send.

Two major components of our platform are the Mininet [26] real-time network emulator and the Ray [28] project. We use Mininet to deploy a virtual network topology and Ray to integrate RL algorithms with the emulator. While Mininet is entirely virtual and is limited in its ability to generalize DC traffic, it is capable of approximating real traffic behavior. Mininet has been effectively used as a platform for larger emulation frameworks [14, 35].

3.1 Defining the Environment State

We have opted for a centralized DC management strategy. All RL agents operate on the global view of the network state.

Iroko deploys monitors that collect statistics from switches in the network and store them as a $d \times n$ traffic matrix. This matrix models the data center as a list of n ports with d network features. The agent only uses switch buffer occupancy, interface utilization, and active flows as the environment state[3]. This matrix can be updated on the scale of milliseconds, which is sufficient to sample the majority of DC flows [38, 12].

3.2 The Agent Actions

Our control scheme specifies the percentage of the maximum allowed bandwidth each host can send. We represent this action set as a vector $\vec{a} \in \mathbb{R}^n$ of dimensions equal to the number of host interfaces.

Each dimension a_i represents the percentage[4] of maximum bandwidth allocated to the corresponding host by the following operation:

$$\text{bw}_i \leftarrow bw_{max} * a_i \quad \forall \quad \text{i} \in \text{hosts} \tag{1}$$

Similar to FastPass [34], the granularity of this allocation scheme can be extended to a per-flow allocation, with a minimum bandwidth guarantee per host. For now, we leave it to the agents to estimate the best percentage allocation according to the reward. In an ideal instantiation of a DC under this system, packet-loss will only rarely, if ever, occur, and will minimally impact the network utilization.

3.3 Congestion Feedback Reward Function

Choosing an appropriate reward function is crucial for the agent to learn an optimal policy. Inappropriately defined reward can lead to unexpected behavior [27]. The Iroko emulator allows the definition of arbitrary reward functions based on the provided input state. In our initial setup, we have decided to minimize switch bufferbloat [17]. The goal is to reduce the occurrence of queuing on switch interfaces, as it indicate congestion and inefficient flow distribution.

We follow a common trade-off model which is inspired by recent work on TCP CC optimization [45]:

$$R \leftarrow \sum_{i \in hosts} \underbrace{bw_i/bw_{max}}_{\text{bandwidth reward}} - \underbrace{\text{ifaces}}_{weight} \cdot \underbrace{(q_i/q_{max})^2}_{\text{queue penalty}} - \underbrace{\text{std}}_{devpenalty} \tag{2}$$

This equation encourages the agent to find an optimal, fair bandwidth allocation for each host while minimizing switch queues. In the equation, *bandwidth reward* is the current network utilization and is the only positive reward, while *queue penalty* is the current queue size of switch ports weighted by the number of interfaces. The *dev penalty* penalizes for actions with high standard deviation to ensure allocation fairness and to mitigate host starvation.

4 Preliminary Experiments

As preliminary analysis we used Iroko to compare the performance of three established deep RL algorithms: REINFORCE [53, 48], the Proximal Policy Gradient (PPO) [43], and Deep Deterministic Policy Gradient (DDPG) [29].

We use the RLlib implementations of the three deep RL algorithms. The library is still growing, so for REINFORCE and PPO we used the default configurations. For DDPG, we chose the parameters to align closely with the original work configuration, except for adding batch normalization. We flatten the collected state into a fully connected neural network architecture (this is an approach similar to [12] and [30]). Although choosing appropriate hyper-parameters can drastically affect algorithm performance [20], we leave tuning to future work. The full configuration details of the algorithms and hardware specifications of our setup are listed in the Appendix.

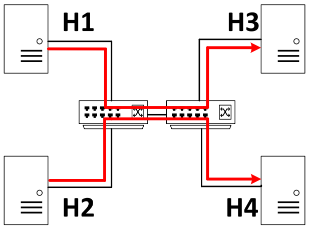
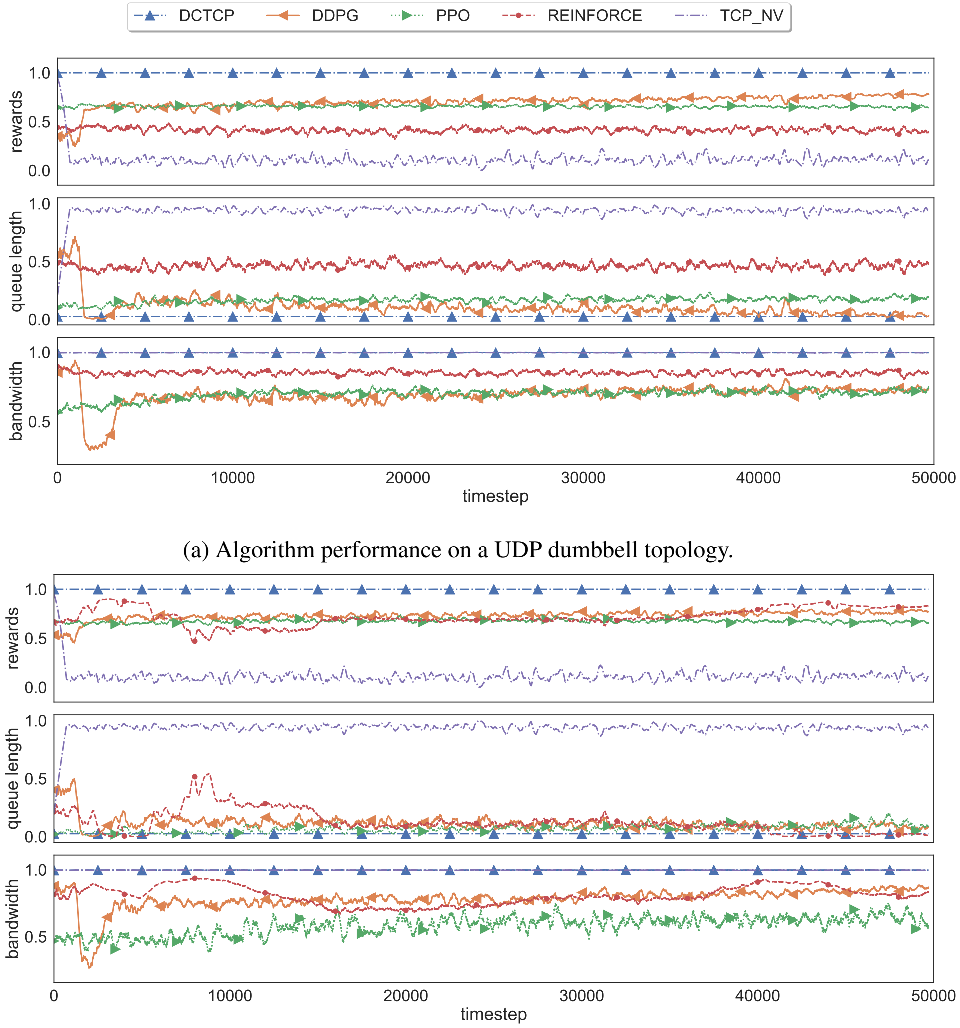


Figure 3: The dumbbell scenario.

Our first benchmark uses a dumbbell topology with four hosts connected over two switches and a single 10 Mbit link (Figure 3). Hosts `H1` and `H2` are sending

[3] We presume that switches in a real DC can reliably provide these statistics.

[4] It is the user's responsibility to squash these values to the appropriate range of 0 to 1.0.



(a) Algorithm performance on a UDP dumbbell topology.

(b) Algorithm performance on a TCP dumbbell topology.

constant 10 Mbit traffic to the hosts on the opposite side, causing congestion on the central link. A trivial and fair solution to this scenario is an allocation of 5 Mbit for each host pair. As comparison, we run a separate RL environment where all CC decisions are strictly managed by TCP. We compare the RL policies against TCP New Vegas [44] and Data Center TCP (DCTCP) [4].

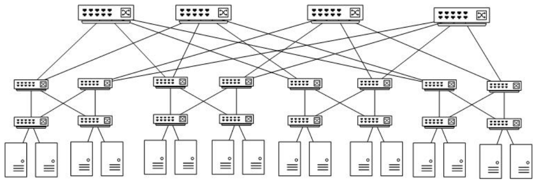


Figure 4: The fat-tree scenario.

TCP New Vegas and DCTCP are state-of-the-art congestion avoidance algorithms optimized for DC traffic. DCTCP requires Active Queue Management (AQM) [6] on switches, which marks packets exceeding a specific queue threshold with an explicit congestion notification (ECN) tag before delivering them. DCTCP uses this information to adjust its sending rate preemptively. While DCTCP is effective at avoiding congestion and queue build-up, it is still incapable of avoiding queues or bursts altogether [51, 54]. In our experiments we treat DCTCP as the possible TCP optimum and TCP New Vegas as a conventional TCP baseline.

For the TCP algorithms the RL policy is ignored, but the same reward function is recorded. This serves as a baseline comparison, and provides empirical evidence on the behavior of a classical CC scheme viewed through the lens of a reinforcement policy.

We run each RL policy five times for 50,000 timesteps using both TCP and UDP as transport protocols. Each timestep is set to 0.5 seconds to give enough time to collect the change in queues and bandwidth in the network. A full test under this stepsize translates to a duration of ~7 hours per test.

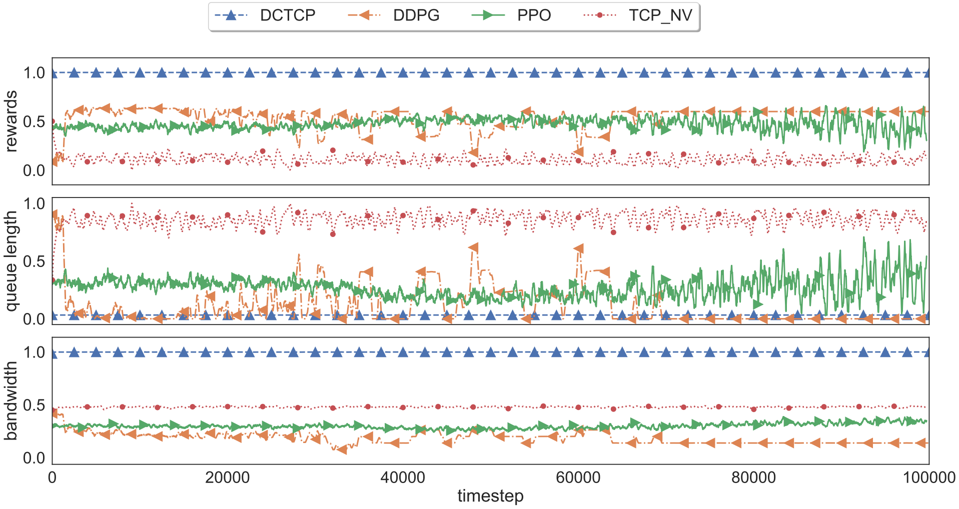


Figure 5: Algorithm performance on a fat-tree topology.

TCP's flow control acts as a decentralized CC agent, which is a potential factor in confounding the contribution of the policy learned by the RL algorithm. UDP does not have flow control and is not typically used in the DC setting, but this pushes all congestion management to the RL algorithm. We measure the change in average reward, the utilization of each host interface, and the queue build-up on the congested link.

In addition, we investigate PPO's and DDPG's performance in a more complex DC scenario. We run the each algorithm for 100,000 timesteps (14 hours) on a UDP-based fat-tree topology [1](Fig. 4 with 16 hosts and 20 switches. This results in a 80×16 state matrix and an action vector of length 16.

4.1 Results

Figures 2a and 2b plot the dumbbell topology results for the UDP and TCP settings. We see that DDPG achieves the highest reward and continues to improve. All algorithms beat TCP New Vegas in reward, while minimizing the queue buildup on the congested link. This implies that the algorithms quickly learn a positive allocation. Interestingly, REINFORCE performs much better in combination with TCP. Policies such as PPO or REINFORCE are estimated to work better in stochastic environments [20]. DC environments, and TCP in particular, exhibit stochastic characteristics (e.g., wildly varying throughput or unstable flow behavior), which may explain the good performance of REINFORCE.

DDPG emerges as the best choice of the three implementations. This is likely due to the deterministic nature of our traffic. With a more stochastic pattern we expect to see a shift in performance in favor of PPO. We leave this as future work.

Figures 5 shows the fat-tree measurements. DDPG has the best performance, but converges to a very low bandwidth setting. PPO is very volatile but continuously improves in bandwidth. We believe a higher step count, multiple runs, and configuration tuning are required to produce conclusive evidence on the algorithms' performance.

Overall, however, DCTCP remains unbeaten. This is expected as DCTCP is a highly optimized algorithm with continuous kernel support. Our reinforcement learning algorithms only use a basic configurations and perform actions on a coarse 0.5 second scale. In addition to reducing the action granularity, we are also investigating solutions that allow for more complex actions (e.g., providing a series of actions for the next n-seconds).

5 Concluding Remarks

Deploying reinforcement learning in the DC remains challenging. The tolerance for error is low and decisions have to be made on a millisecond scale. Compared to a TCP algorithm on a local host, a DC agent has to cope with significant delay in its actions. The chaotic and opaque nature of DC networks makes appropriately crediting actions nearly impossible. Rewards, actions, and state can be mix-and-matched arbitrarily. There is no indication or theoretical insight if a particular combination will be successful. The fact that traffic has to be evaluated in real-time leads to slow prototyping and agent learning curve. Optimizing a network of a mere 16 hosts is already a substantial task, since each node is an independent actor with unpredictable behavior.

Nonetheless, our initial results are encouraging. In the dumbbell tests, the agents can quickly learn a fair distribution policy, despite the volatility of the network traffic. DDPG and PPO even exceed the TCP New Vegas baseline and demonstrate steady improvement.

We plan to continue work on our benchmarking tool and focus on improving the emulator performance for fat-tree scenarios. This includes hyper-parameter tuning and deployment automation using the Ray framework. We are looking into using meta-information such as job deployments, bandwidth requests by nodes, or traffic traces as additional state information. We also plan to extend the range of reward models, topologies, traffic patterns, and algorithms to truly evaluate the performance of reinforcement learning policies. Iroko is an open-source project available at `https://github.com/dcgym/iroko`.

Acknowledgments

We like to thank Mark Schmidt for his feedback on this paper. This work is supported by Huawei and the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

References

[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. 63–74.

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA* (2010), pp. 281–296.

[3] Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V. T., Matus, F., Pan, R., Yadav, N., and Varghese, G. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 503–514.

[4] 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* (New York, NY, USA, 2010), SIGCOMM '10, ACM, pp. 63–74.

[5] Arun, V., and Balakrishnan, H. Copa: Practical delay-based congestion control for the internet. In *Proceedings of the Applied Networking Research Workshop* (New York, NY, USA, 2018), ANRW '18, ACM, pp. 19–19.

[6] Athuraliya, S., Low, S. H., Li, V. H., and Yin, Q. Rem: Active queue management. *IEEE Network 15*, 3 (May 2001), 48–53.

[7] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 267–280.

[8] Benson, T., Anand, A., Akella, A., and Zhang, M. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, ACM, pp. 8:1–8:12.

[9] Boyan, J. A., and Littman, M. L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the 6th International Conference on Neural Information Processing Systems* (San Francisco, CA, USA, 1993), NIPS'93, Morgan Kaufmann Publishers Inc., pp. 671–678.

[10] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *CoRR abs/1606.01540* (2016).

[11] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. Bbr: Congestion-based congestion control. *Queue 14*, 5 (Oct. 2016), 50:20–50:53.

[12] Chen, L., Lingys, J., Chen, K., and Liu, F. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2018), SIGCOMM '18, ACM, pp. 191–205.

[13] Cho, I., Jang, K., and Han, D. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, ACM, pp. 239–252.

[14] Csoma, A., Sonkoly, B., Csikor, L., Németh, F., Gulyas, A., Tavernier, W., and Sahhaf, S. Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 125–126.

[15] Dong, M., Li, Q., Zarchy, D., Godfrey, B., and Schapira, M. Rethinking congestion control architecture: Performance-oriented congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 365–366.

[16] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, pp. 1329–1338.

[17] Gettys, J., and Nichols, K. Bufferbloat: Dark buffers in the internet. vol. 9, ACM, pp. 40:40–40:54.

[18] Grosvenor, M. P., Schwarzkopf, M., Gog, I., Watson, R. N. M., Moore, A. W., Hand, S., and Crowcroft, J. Queues don't matter when you can jump them! In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2015), NSDI'15, USENIX Association, pp. 1–14.

[19] Hemminger, S. Network emulation with netem. In *linux.conf.au - Australia's National Linux Conference* (2005), pp. 18–23.

[20] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (2018), pp. 3207–3214.

[21] Jose, L., Yan, L., Alizadeh, M., Varghese, G., McKeown, N., and Katti, S. High speed networks need proactive congestion control. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2015), HotNets-XIV, ACM, pp. 14:1–14:7.

[22] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. The nature of data center traffic: Measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2009), IMC '09, ACM, pp. 202–208.

[23] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014).

[24] Kober, J., and Peters, J. *Reinforcement Learning in Robotics: A Survey*. Springer International Publishing, Cham, 2014, pp. 9–67.

[25] Lanctot, M., Zambaldi, V. F., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., and Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. *CoRR abs/1711.00832* (2017).

[26] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets-IX, ACM, pp. 19:1–19:6.

[27] Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. AI safety gridworlds. *CoRR abs/1711.09883* (2017).

[28] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning* (Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 3053–3062.

[29] 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. *CoRR abs/1509.02971* (2015).

[30] Mao, H., Alizadeh, M., Menache, I., and Kandula, S. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2016), HotNets '16, ACM, pp. 50–56.

[31] Marivate, V. N. *Improved empirical methods in reinforcement-learning evaluation*. PhD thesis, 2015.

[32] Mirza, M., Sommers, J., Barford, P., and Zhu, X. A machine learning approach to tcp throughput prediction. *IEEE/ACM Trans. Netw. 18*, 4 (Aug. 2010), 1026–1039.

[33] 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. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 537–550.

[34] Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D., and Fugal, H. Fastpass: A centralized zero-queue datacenter network. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 307–318.

[35] Peuster, M., Karl, H., and van Rossem, S. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (Nov 2016), pp. 148–153.

[36] Pfaff, B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2015), NSDI'15, USENIX Association, pp. 117–130.

[37] Raghu, M., Irpan, A., Andreas, J., Kleinberg, R., Le, Q. V., and Kleinberg, J. M. Can deep reinforcement learning solve erdos-selfridge-spencer games? *CoRR abs/1711.02301* (2017).

[38] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the social network's (datacenter) network. vol. 45, ACM, pp. 123–137.

[39] Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. Deep reinforcement learning framework for autonomous driving. *CoRR abs/1704.02532* (2017).

[40] Schapira, M., and Winstein, K. Congestion-control throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2017), HotNets-XVI, ACM, pp. 122–128.

[41] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *CoRR abs/1511.05952* (2015).

[42] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *CoRR abs/1506.02438* (2015).

[43] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (2017).

[44] Sing, J., and Soh, B. Tcp new vegas: improving the performance of tcp vegas over high latency links. In *Network Computing and Applications, Fourth IEEE International Symposium on* (2005), IEEE, pp. 73–82.

[45] Sivaraman, A., Winstein, K., Thaker, P., and Balakrishnan, H. An experimental study of the learnability of congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 479–490.

[46] STREIFFER, C., CHEN, H., BENSON, T., AND KADAV, A. Deepconfig: Automating data center network topologies management with machine learning. *CoRR abs/1712.03890* (2017).

[47] SUTTON, R. S., AND BARTO, A. G. *Introduction to Reinforcement Learning*, 1st ed. MIT Press, Cambridge, MA, USA, 1998.

[48] SUTTON, R. S., MCALLESTER, D., SINGH, S., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 1999), NIPS'99, MIT Press, pp. 1057–1063.

[49] TESAURO, G., DAS, R., CHAN, H., KEPHART, J., LEVINE, D., RAWSON, F., AND LEFURGY, C. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 1497–1504.

[50] VALADARSKY, A., SCHAPIRA, M., SHAHAF, D., AND TAMAR, A. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2017), HotNets-XVI, ACM, pp. 185–191.

[51] VAMANAN, B., HASAN, J., AND VIJAYKUMAR, T. Deadline-aware datacenter tcp (d2tcp). *SIGCOMM Comput. Commun. Rev. 42*, 4 (Aug. 2012), 115–126.

[52] WHITESON, S., TANNER, B., TAYLOR, M. E., AND STONE, P. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)* (April 2011), pp. 120–127.

[53] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 229–256.

[54] WILSON, C., BALLANI, H., KARAGIANNIS, T., AND ROWTRON, A. Better never than late: Meeting deadlines in datacenter networks. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 50–61.

[55] WINSTEIN, K., AND BALAKRISHNAN, H. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 123–134.

[56] YAN, F. Y., MA, J., HILL, G. D., RAGHAVAN, D., WAHBY, R. S., LEVIS, P., AND WINSTEIN, K. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, 2018), USENIX Association, pp. 731–743.

[57] ZAKI, Y., PÖTSCH, T., CHEN, J., SUBRAMANIAN, L., AND GÖRG, C. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 509–522.

[58] ZHANG, C., VINYALS, O., MUNOS, R., AND BENGIO, S. A study on overfitting in deep reinforcement learning. *CoRR abs/1804.06893* (2018).

6 Appendix

All experiments were run using a Linux 4.15.0-34 kernel on a single 8 core (2xIntel Xeon E5-2407) machine with 32GB of RAM. We used Ray version 0.5.3. Network emulation is performed using the Linux NetEm [19] package. All hosts are connected over instances of the Open vSwitch [36]. The sending rate of hosts is adjusted via the following Linux traffic control command:

```
tc qdisc change dev [iface] root fq maxrate [bw]mbit
```

The monitors collect network statistics using the Linux tools `ifstat`, `tc qdisc show`, or `tcpdump`. Traffic is generated using the Goben traffic generator written in Go. Goben is open-source and located at `https://github.com/udhos/goben`. Each algorithm utilize a neural network model with two hidden layers both with 256 neurons, and `tanh` for activation. The DDPG algorithm also uses this model with additional parameters for the actor and critic neural networks as specified in the table. Hyperparameter names are written to closely follow the current variable named used in RLlib:

Hyperparameter	Value
DDPG	
θ	0.15
σ	0.2
Noise scaling	1.0
Target network update frequency	Every update
τ	10^{-3}
Use Prioritized Replay Buffer [41]	False
Actor hidden layer sizes	400, 300
Actor activation function	ReLU
Critic hidden layers sizes	400, 300
Critic activation function	ReLU
Optimizer	Adam [23]
Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Weight decay coefficient	10^{-2}
Critic loss function	Square loss
PPO	
Use GAE [42]	True
GAE Lamda	1.0
KL coefficient	0.2
Train batch size	4000
Mini batch size	128
Num SGD Iterations	30
Optimizer	Adam [23]
Learning rate	$5 * 10^{-5}$
Value function coefficient	1.0
Entropy coefficient	0.0
Clip parameter	0.3
Target Value for KL	0.01
REINFORCE	
Learning rate	10^{-4}
Optimizer	Adam [23]

Table 1: Configurations for all algorithms.