A Deep Reinforcement Learning Perspective on Internet Congestion Control

Nathan Jay [*1] **Noga H. Rotman** [*2] **P. Brighten Godfrey** [1] **Michael Schapira** [2] **Aviv Tamar** [3]

Abstract

We present and investigate a novel and timely application domain for deep reinforcement learning (RL): Internet congestion control. Congestion control is the core networking task of modulating traffic sources' data-transmission rates to efficiently utilize network capacity, and is the subject of extensive attention in light of the advent of Internet services such as live video, virtual reality, Internet-of-Things, and more. We show that casting congestion control as RL enables training deep network policies that capture intricate patterns in data traffic and network conditions, and leverage this to outperform the state-of-the-art. We also highlight significant challenges facing real-world adoption of RL-based congestion control, including fairness, safety, and generalization, which are not trivial to address within conventional RL formalism. To facilitate further research and reproducibility of our results, we present a test suite for RL-guided congestion control based on the OpenAI Gym interface.

1. Introduction

Deep RL has gained substantial popularity in light of its applicability to real-world decision making, generating headlines by facilitating complex tasks like protein folding (R.Evans, 2018) and beating human experts in challenging contexts such as Go (Silver et al., 2016). We wish to capitalize on these advancements to tackle the crucial and timely challenge of Internet congestion control.

1.1. Internet Congestion Control

In today's Internet, multiple network users contend over scarce communication resources. Consequently, the data-transmission rates of different traffic sources must be modulated dynamically to efficiently utilize network resources and provide good user experience. This challenge is termed "congestion control" and is fundamental to computer networking research and practice. Indeed, congestion control has a crucial impact on user experience for Internet services such as video streaming and voice-over-IP, as well as emerging Internet services such as augmented and virtual reality, Internet of Things, edge computing, and more.

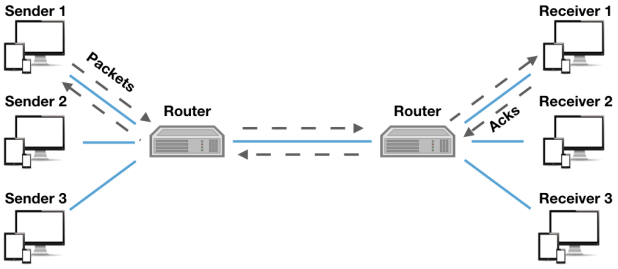


Figure 1: Multiple traffic flows sharing a link

Consider the illustrative example in Figure 1, which depicts multiple *connections* (also referred to as "*flows*") sharing a *single* communication link. Each connection consists of a traffic sender and a traffic receiver. The sender streams data packets to the receiver and constantly experiences feedback from the receiver for sent packets in the form of packet acknowledgements (ACKs). The sender adjusts its transmission rate in response to this feedback. The manner in which the sending rate is adjusted is determined by the *congestion control protocol* employed by the connection's two endpoints. The interaction of different connections gives rise to network dynamics, derived from the connections' congestion control protocols, the link's capacity (bandwidth), and also the link's buffer size and *packet-queueing policy*, which determine how (and whose) excess traffic is discarded.

Even this simple single-link scenario illustrates the complexity of congestion control, in which different connections select rates in an uncoordinated, decentralized manner, typically with no explicit information about competing connections (number, times of entry/exit, employed congestion control protocols), or the network (link's bandwidth, buffer size, and packet-queuing policy). Naturally, these challenges are

[*]Equal contribution [1]University of Illinois at Urbana-Champaign [2]Hebrew University of Jerusalem [3]Technion. Correspondence to: Nathan Jay <njay2@illinois.edu>, Noga H. Rotman <nogar02@cs.huji.ac.il>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

exacerbated when traffic is forwarded across multiple links, and since networks greatly vary in sizes, link capacities, network latency, level of competition between connections, and more. Consequently, even after three decades of research on Internet congestion control, heated debates linger about the "right" approaches (Schapira & Winstein, 2017).

1.2. Motivating RL-Guided Congestion Control

We argue that investigating RL in the context of congestion control is important in two respects: (1) improving the performance of a crucial component of the Internet's communication infrastructure, with the potential of impacting the user experience of essentially every performance-sensitive Internet service, and (2) providing an exciting new "playground" for RL schemes that poses novel real-world-motivated research challenges.

A congestion control protocol can be regarded as mapping a locally-perceived history of feedback from the receiver, which reflects past traffic and network conditions, to the next choice of sending rate. We hypothesize that such local history contains information about patterns in traffic and network conditions that can be exploited for better rate selection by *learning* the mapping from experience via a deep RL approach. This hypothesis is motivated by the recent success of deep learning in extracting useful features across various domains, including image, speech, and games.

To illustrate the types of patterns that can be learned by RL schemes, consider the following two examples:

Distinguishing non-congestion loss from congestion-induced loss. A packet loss can be indicative of different phenomena (Dong et al., 2015; Cardwell et al., 2016; Dong et al., 2018); some might be induced by congestion, i.e., result from exceeding the network capacity, whereas others might be non-congestion-related, e.g., due to handover between mobile base stations. A well-known deficiency of today's prevalent congestion control protocol, the Transmission Control Protocol (TCP), is its innate inability to distinguish between these two forms of packet loss, which can lead to highly suboptimal rate choices (Dong et al. 2015).

Figure 2 plots the throughput of a single flow on a link of bandwidth 30 Mbps, where 1% of sent packets are randomly dropped, for two congestion control protocols: the CUBIC (Ha et al., 2008) variant of TCP, which is Linux's default; and a protocol designed within our RL framework.

Observe that TCP CUBIC, which halves the sending rate upon *any* occurrence of loss, fails to fully utilize the link's bandwidth. In contrast, the RL-based protocol effectively distinguishes between the two types of losses; it often increases its rate upon encountering randomly-generated packet losses but avoids overshooting the link's capacity by "too much" by not increasing its rate when losses are due to congesting the link. Thus, the RL-based protocol is capable of maintaining high link utilization. Intuitively, distinguishing between these two types of losses is possible since random loss is unaffected by the sender's actions, whereas congestion-induced loss rate increases with the sending rate.

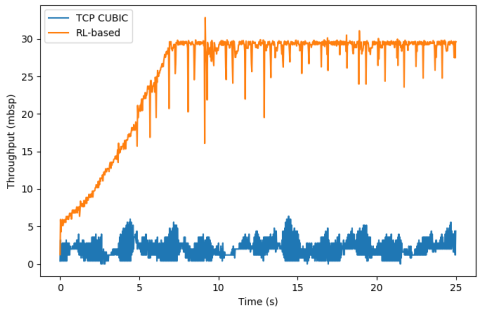


Figure 2: A 25-second trace of throughput for TCP-CUBIC and an RL-based protocol with 1% random loss on a 30 Mbps bandwidth link and a 10-packet queue.

Adapting to variable network conditions. Network conditions, such as available link capacities, packet loss rates, and end-to-end latency, might be highly dynamic and change considerably over time (e.g., in mobile/cellular networks, Winstein et al. 2013). TCP is notoriously bad at adapting to variable network conditions (Dong et al., 2015; Cardwell et al., 2016; Dong et al., 2018; Arun & Balakrishnan, 2018).

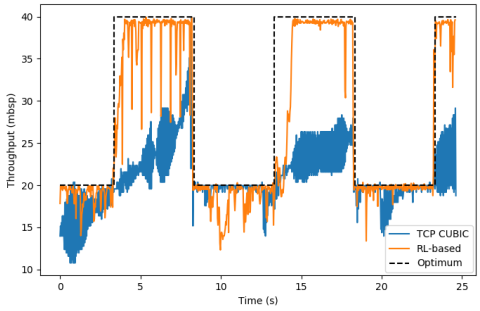


Figure 3: A 25-second trace of the throughput of TCP-CUBIC and an RL-based protocol on a link which alternates between 20 and 40 Mbps every five seconds.

Consider a single flow on a link whose capacity alternates between 20 Mbps and 40 Mbps every 5 seconds (with no random loss). Ideally, the congestion control protocol would change its sending rate so as to closely match the link's bandwidth, as illustrated by the dashed black line in Figure 3, thus utilizing the capacity without causing congestion. As shown in the figure, TCP CUBIC fails to do so. In contrast, a protocol designed within our RL framework closely approximates the desired behavior. Intuitively, this protocol learns, e.g., that sudden steep rises in packet loss indicate

that available bandwidth has decreased considerably, and that no packet loss when the sending rate exceeds 20 Mbps indicates that the available bandwidth is higher (40 Mbps).

1.3. Our Contribution

We formulate a novel framework for RL-based congestion control protocol design, which extends the recently introduced Performance-oriented Congestion Control (PCC) approach (Dong et al., 2015; 2018). We discuss challenges involved in casting congestion control as an RL task. We also describe remaining challenges facing the real-world adoption of RL-based congestion control schemes, such as fairness, safety, and generalization, which are not trivial to address within conventional RL formalism.

We utilize our framework to design Aurora. Aurora employs deep RL (Sutton et al., 1998; Schulman et al., 2015) to generate a policy for mapping observed network statistics (e.g., latency, throughput) to choices of rates. Our preliminary evaluation results suggest training Aurora in simple, simulated environments is sufficient to generate congestion control policies that perform well also in very different network domains and which are comparable to, or outperform, recent state-of-the-art handcrafted protocols.

Our code[1] is open-sourced as an OpenAI Gym environment and an accompanying testing module, to be used by RL researchers and practitioners to evaluate their algorithms.

2. RL Approach to Internet CC

We next provide a high-level overview of RL and explain how congestion control can be formulated as an RL task.

2.1. Background: Reinforcement Learning

In RL (Sutton et al., 1998), an *agent* solves a sequential decision making problem by interacting with an *environment*.

At each discrete time step $t \in 0, 1, ...$, the agent observes a (locally perceptible) state of the environment s_t, and selects an action a_t. At the following time step $t+1$, the agent observes a *reward* r_t, representing its loss/gain after time t, as well as the next state s_{t+1}.[2] The agent's goal is to choose a policy π mapping states to actions that maximize the *expected cumulative discounted return* $R_t = \mathbb{E}\big[\sum_t \gamma^t \cdot r_t\big]$, for $\gamma \in \big[0, 1\big)$. The parameter γ is termed the *discount factor*. For large or continuous state and action spaces, this problem is intractable, and recent advances in deep RL employ deep neural networks to approximate the optimal π (Schulman et al., 2015; Silver et al., 2017).

2.2. Congestion Control as RL

We formulate congestion control as a sequential decision making problem under the RL framework.

Actions are changes to sending rate. In our formulation, the agent is the sender of traffic and its actions translate to changes in sending rates. To formalize this, we adopt the notion of *monitor intervals* (MIs) from (Dong et al., 2015; 2018). Time is divided into consecutive intervals. In the beginning of each MI t, the sender can adjust its sending rate x_t, which then remains fixed throughout the MI. After experimenting with several options, we chose to express actions as *changes* to the current rate (see Section 3.1).[3]

States are bounded histories of network statistics. After the sender selects rate x_t at MI t, it observes the results of sending at that rate and computes a statistics vector v_t from received packet-acknowledgements. We restrict our attention below to *statistics vectors* consisting of the following: (i) latency gradient (Dong et al., 2018), the derivative of latency with respect to time; (ii) latency ratio (Winstein & Balakrishnan, 2013), the ratio of the current MI's mean latency to minimum observed mean latency of any MI in the connection's history; and (iii) sending ratio, the ratio of packets sent to packets acknowledged by the receiver.

Networks greatly vary in terms of available bandwidth, latency, and loss rate. Our choice of elements comprising the statistics vector is intended to improve the generalization of our models by avoiding statistics that are expected to be highly variable across connections for no better reason than variation in link properties (e.g., the absolute value of experienced latency in milliseconds).

The agent's selection of the next rate change is a function of a *fixed-length history* of the above statistics vectors collected from packet acknowledgements sent by the receiver. Considering a bounded-length history, instead of just the most recent statistics, allows our agent to detect trends and changes in network conditions and react more appropriately. Thus, the state at time t, s_t, is defined to be:

$$s_t = (v_{t-(k+d)}, \ldots, v_{t-d}),$$

for a predetermined constant $k > 0$ and a small number d representing the delay between choosing a sending rate and gathering results. We discuss how the length of the history, i.e., k, affects performance in Section 3.3.

Setting rewards. The reward resulting from sending at a certain rate at a certain time may depend on the performance

[1] https://github.com/PCCproject/PCC-RL

[2] In our setting, the next state is drawn from the environment's transition dynamics, taking into account the agent's action, the actions of other agents, and parameters unavailable to the agent, e.g. link capacity. This makes the problem an instance of a partially observable Markov decision process (Kaelbling et al., 1998).

[3] While our action formulation (periodic rate changes) is less nuanced than allowing the sender to choose the exact timing of each packet transmission, such a formulation is too expensive to realize with today's transmission speeds.

requirements of the specific application; some applications (e.g., online gaming) might require very low latency while for others (e.g., large file transfers) high bandwidth is much more crucial; some services might prefer low-but-constant bandwidth (no "jitter"), while others may desire higher bandwidth and be more tolerant to bandwidth variation. We discuss specific reward functions in Section 3.1.

The effects of an action (change in rate) could potentially have non-immediate consequences, e.g., sending at too fast a pace could overload buffers and result in future packet losses and delays. In RL, long-horizon decision making is captured via the discount factor γ. We discuss the impact of γ in our framework in Section 3.3.

2.3. Other Considered Approaches

We considered alternative formulations of congestion control as a learning task (most notably as a bandits problem) and alternative model architectures (including linear models) prior to settling on the ones presented here.

Our results (see Figure 5) show that to learn a reasonable policy, the discount factor γ cannot be too low (e.g., γ should be at least 0.5), with high discount factors ($\gamma = 0.99$) resulting in much faster learning. This is in agreement with the sequential nature of the task, in which rewards might be delayed due to effects such as limited buffer size on the link and increase in latency as a consequence of increase in link occupancy.

In addition, training linear models resulted in much worse performance—noticeably worse than even a single layer neural network. We also tried simple random search and hill-climbing for linear models, whic (Mania et al., 2018) recently showed performs well on continuous Mujoco tasks, but these were not competitive in our context.

In our experiments (Section 4), a discount of 0.99 resulted in much faster learning (though eventually reaching a performance similar to 0.5 - see Figure 5), showing that the stronger signal from delayed rewards is important.

3. Introducing Aurora

In this section we introduce Aurora: a specific implementation of RL for congestion control, based on the formulation above, that achieves state-of-the-art results. Our code is available at our github repo.

3.1. Architecture

RL inputs and outputs. We choose to map our agent's output based on the statistic vectors discussed in Section 2 to a change in sending rate x_{t-1} according to:

$$x_t = \begin{cases} x_{t-1} * (1 + \alpha a_t) & a_t \geq 0 \\ x_{t-1}/(1 - \alpha a_t) & a_t < 0 \end{cases}$$

where α is a scaling factor used to dampen oscillations (we use $\alpha = 0.025$).

Neural network. Neural network architectures vary significantly and research suggests new architectures at an incredible rate, so choosing *the* optimal architecture is impractical. We show, however, that even a simple architecture, i.e., a small *fully connected* neural network, produces good results. We tested several options for the number of hidden layers and number of neurons per layer and chose an architecture with two hidden layers composed of $32 \rightarrow 16$ neurons and tanh nonlinearity. After training three replicas of each considered architecture, this one produced the highest average training reward and exhibited high performance throughout our evaluation process (see Section 4).

Reward function. We trained Aurora with a linear reward function that rewards throughput while penalizing loss and latency. State-of-the-art PCC-Vivace (Dong et al., 2018) and Copa (Arun & Balakrishnan, 2018) try to optimize reward functions with different exponents and logarithms of these components, but have similar goals (high throughput, low latency, with PCC-Vivace penalizing loss as well). We choose the following linear function instead:

$$10 * throughput - 1000 * latency - 2000 * loss$$

where $throughput$ is measured in packets per second, $latency$ in seconds, and $loss$ is the proportion of all packets sent but not acknowledged. The scale of each factor was chosen to force models to balance throughput and latency for our chosen training parameters. In Section 4 we discuss the objective functions (or lack thereof) for other algorithms where we demonstrate Aurora's throughput-latency tradeoff.

3.2. Training

We train our agent in an open-source gym environment described in detail in Section 5. This environment simulates network links with a range of parameters. Our model was trained using the PPO algorithm (Schulman et al., 2017), as implemented in the stable-baselines python package (based on Dhariwal et al. 2017).

3.3. Choice of Parameters

While many parameters affect the quality of our final model, we next discuss two significant parameter choices: history length, and discount factor.

History length. A history length of k means that the agent makes a decision based on the k latest MIs worth of data. Intuitively, increasing history length should increase performance, as extra information is given. We trained models with k ranging from 1 to 10 MIs. Figure 4 shows the training reward of these models. Eventually, a model with $k = 2$ was comparable to the model with $k = 10$, but a model with a single history does not learn a comparable

policy in our chosen number of training episodes.

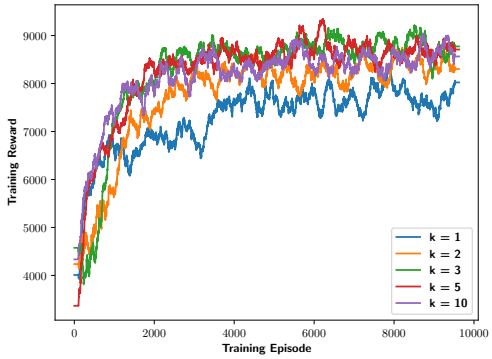


Figure 4: Training reward for agents with different values of history length. Lines are smoothed averages of three models.

Discount factor. We examined three different values of γ and determined that $\gamma = 0.99$ gave the best results quickly, while $\gamma = 0.50$ eventually learned a reasonable policy, and $\gamma = 0.00$ failed to learn a useful policy. This seems intuitive, given the gap between taking an action and observing a reward. Figure 5 shows these trends.

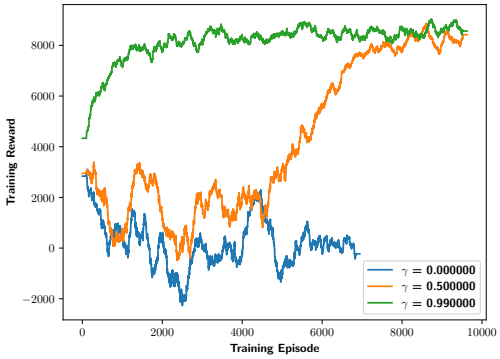


Figure 5: Training reward for agents with different values of gamma. Each line is the smoothed average of three models. The line for $\gamma = 0.00$ ends early because training exceeded a budgeted time.

4. Evaluation

Our training framework uses a simple simulation of a single traffic source on network links with varied parameters. Our testing suite, however, goes far beyond that—including link parameters outside the scope of the training values, dynamic links as opposed to the purely static links in training, and training in an emulated rather than simulated environment. In particular, we test the performance and robustness of our trained model using standard network research tools (Mininet (Fontes et al., 2015) and Pantheon (Yan et al., 2018)) for a wide variety of network scenarios, and provide comparisons to state-of-the-art congestion control protocols.

4.1. Robustness

Figure 6 demonstrates our model's behavior when bandwidth, latency, queue size and random loss rate vary far outside of our training conditions. The Pantheon project showed that varying these four link parameters is sufficient to emulate a wide variety of real-world network conditions. (Pantheon also includes a fifth parameter, setting queue behavior to fixed or exponentially-distributed service times, to better emulate mobile LTE links; this parameter is not available in Mininet.) We run each test for two minutes with a single sender over a single link. For each test, we compare the results of our model against TCP CUBIC (the current standard for congestion control), and PCC Vivace (Dong et al. 2018, a robust, state-of-the-art algorithm). Our supplemental material includes comparisons to several other state-of-the-art algorithms. Unless otherwise stated, Mininet links are configured with 30 Mbps capacity, 30 ms of latency, a 1000-packet queue, and 0% random loss.

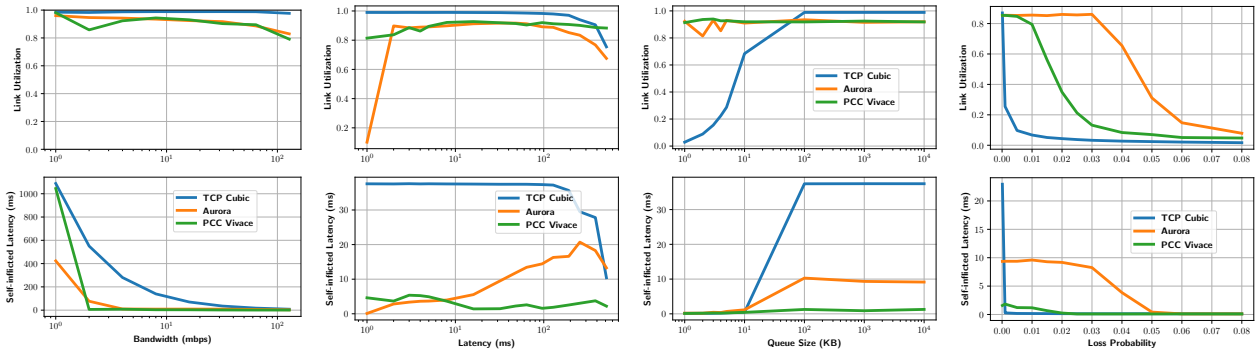
Bandwidth sensitivity. Our model was trained on links with bandwidth between 1.2 Mbps and 6 Mbps, but any reasonable congestion control should operate in a much wider range. In Figure 6a we show that Aurora operates well between 1 Mbps and 128 Mbps, more than 20× above its training environment. We start from our standard link configuration and configure the bandwidth differently for each test, ranging from 1 to 128 Mbps. Our model achieves near-perfect link utilization at all capacities, with latency far below TCP, and comparable to PCC-Vivace.

Latency sensitivity. We trained Aurora with link latency varying between 50ms and 500ms, but many Internet connections are expected to have latency much lower (and rarely higher) than this range. To test the robustness of our model on links with different latency, we vary the latency between 1 ms and 512 ms. Aurora performs poorly at 1ms latency because our emulation environment includes real packet processing, a process that adds noise on the order of around 1ms to the latency, but which was not present in training.

Queue sensitivity. Aurora was trained on links with queue size varying between 2 and 2981 packets. For this set of tests, we varied queue size between 1 and 10,000 packets. Figure 6c shows our model has throughput comparable to PCC-Vivace at all queue sizes and provided significant throughput improvements over TCP CUBIC in that range.

Loss sensitivity. We trained Aurora with random loss probability varying between 0% and 5%. This range already captures the full range of expected conditions for most Internet traffic, but we test up to 8% random loss probability to explore the robustness of our model. In this test, Aurora provides near-capacity throughput at higher loss than either of the other algorithms.

Dynamic links. Some network links, particularly mobile links, have variable capacity that can depend on real-time



(a) Bandwidth sensitivity (b) Latency sensitivity (c) Buffer size sensitivity (d) Loss rate sensitivity

Figure 6: Tests over a single link, showcasing the model's sensitivity to changes in bandwidth, latency queue size and loss rate. Throughput and self-inflicted latency are shown as the relevant parameter varies. In the top graph, higher is better. In the bottom graph, lower is better.

changes in signal strength, whether from interference, distance or surroundings. The exact dynamics of link capacity vary widely, so we examine a simplified case. In this scenario, we consider a link whose capacity varies every five seconds to a newly chosen value distributed between 16 Mbps and 32 Mbps uniformly at random. We allow Aurora, along with five baseline schemes, to run at least five times on this dynamic link setup for two minutes per run.

Figure 7 shows as expected, the schemes have different tradeoff points between latency and throughput. TCP CUBIC achieves high throughput at the expense of significantly higher latency and well-documented bad performance in a broad range of network environments (see Dong et al. 2015 and references therein). Aurora achieves throughput nearly matching BBR (Cardwell et al., 2016) but with significantly better latency. Recent designs have worked to achieve lower latency to better support applications like small web queries, real-time video, and gaming. Among the lower-latency schemes, Aurora achieves both higher throughput and lower latency than PCC-Vivace on average, and higher throughput with similar average latency to RemyCC (Winstein & Balakrishnan, 2013), making it preferable on these dynamic links. Finally, Aurora comes close to Copa's performance with 4.2% better throughput and 16% worse latency.

4.2. Evaluation Summary

Our evaluation results demonstrated two key conclusions. First, Aurora is surprisingly robust to environments outside the scope of its training. Second, Aurora is comparable to or outperforms the state-of-the-art in congestion control (BBR, PCC-Vivace, RemyCC, and Copa). This suggests that applying deep RL to congestion control, even with a fairly simple neural network architecture and with fairly limited training, has the potential of outperforming carefully-handcrafted protocols.

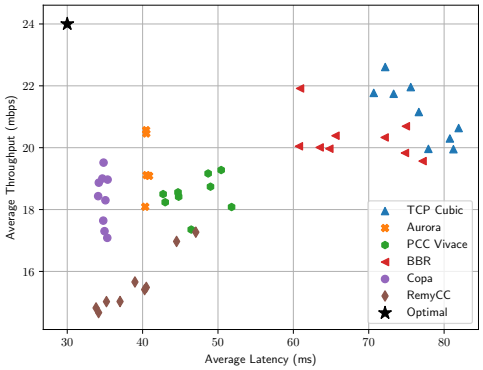


Figure 7: Average throughput and latency on a dynamic link with 32ms of base latency, a 500 packet queue and no random loss.

5. A Testbed for RL Congestion Control

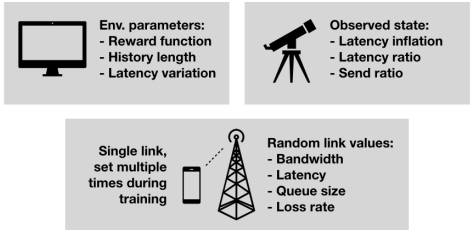


Figure 8: A high-level look at our custom training framework, implemented as an OpenAI Gym environment. By randomizing the link properties during a run, it provides diversified training scenarios.

Existing network simulators incorporate complex aspects of the network including reachability, routing, packet header parsing, among many other factors that may be useful for other domains, but are irrelevant for congestion control.

These extra features complicate and slow down the process of training a congestion control algorithm. To train Aurora we implemented a novel simulator, which realistically mimics Internet links with various characteristics. This simulator is sufficiently accurate that models trained in our simulator perform well in the emulated environment used in Section 4, which sends real packets using a real Linux stack across virtual network interfaces using standard networking tools.

To open our Internet congestion control environment to the machine learning community in the most standard and accessible way, we provide an OpenAI Gym environment based on our in-house simulator, whose entire code base fits in a few hundred lines of pure Python. This section covers the basic design and the three abstractions (*links*, *packets* and *senders*) that we use to create a simple simulated network. For a complete description of the environment with source code and documentation, see our github repo.

Bandwidth	Latency	Queue size	Loss rate
100-500 pps	50-500ms	2-2981 packets	0-5%

Table 1: Training framework variables' ranges, all drawn uniform-randomly, except for the queue size, for which the log is drawn uniform-randomly.

5.1. Links

We simulate links as FIFO queues with the four key parameters identified in Pantheon: *bandwidth*, *latency*, *random loss rate* and *queue size*, each serving the same purpose as in real computer networks. Table 1 provides the range of values for these parameters in our example setup.

5.2. Packets

Packets normally contain all of the routing information and actual data of network communications, but we don't care about the data, and our routes are vastly simplified. To us, a packet is just a tuple of <sender, latency, is_dropped>. The sender tells us the route, the latency tells us what the sender will observe when an acknowledgement is returned, and is_dropped tells us if we should return an acknowledgement.

5.3. Senders

Senders are the agents in our networks, each with a *rate*, *route* and list of observations. A sender's rate determines how fast packets are sent on the sender's *route*, a list of links that data and acknowledgements traverse. This route is fixed at network creation time, a reasonably realistic situation, as Internet routes change orders of magnitude more slowly than congestion control decisions are made. Senders then make observations that are given to our machine learning agent. Figure 8 gives a list of the observations made by our sender, chosen to include meaningful values used in previous congestion control algorithms (Winstein & Balakrishnan, 2013; Dong et al., 2018).

Using our three objects to create a complete network in our environment is straightforward. The example gym environment in this paper uses just two Links (think two lanes of a street) and single Sender object. The links are initialized with bounded random values for bandwidth, latency, queue size and loss rate given in Table 1. The Sender is given an initial sending rate between 30% and 150% of the link bandwidth, and begins taking actions.

6. Remaining Challenges

Our results in Section 4 show that deep RL is indeed capable of learning useful congestion control strategies, which are competitive with dedicated congestion control algorithms. Alongside these promising results, we discuss several aspects of the congestion control problem that are not directly addressed by our RL formulation, and are important for future research. We hope that our OpenAI Gym environment would encourage research along these directions.

6.1. Fairness

On the Internet, many different congestion control protocols interact, and ours has a significant probability of behaving unfairly in some scenarios (we used neither global optimization as in Remy (Winstein & Balakrishnan, 2013), nor game-theoretic equilibrium analyses as in (Dong et al., 2015; 2018)). However, our agent might not be friendly towards legacy TCP. Worse yet, if it were trained in an environment where it competed with TCP, it might learn to occasionally cause packet loss just to force TCP into backing off and freeing up network capacity. Can our RL agent be trained to "play well" with other protocols (TCP, PCC, BBR, Copa)?

6.2. Multiple Objectives

As mentioned earlier, the objective in congestion control is a topic of debate, and users may prefer different tradeoffs between minimizing latency, maximizing throughput, and other performance criteria. Multiobjective RL (Vamplew et al., 2011) is a framework for handling the tradeoffs between different reward factors, which in principle could be applied here. We are, however, unaware of any recent deep RL studies in this direction.

6.3. Generalization and Adaptation to Changes in the Network

Our results were obtained after training Aurora on specific network conditions, and we showed that our selection of training network conditions resulted in a congestion control

strategy that performed well across a broad range of test conditions. While these empirical results are promising, the question of generalization, or transfer, in RL is an active field of research (Tamar et al., 2016; Barreto et al., 2017; Narasimhan et al., 2017; Higgins et al., 2017), and providing guarantees about the performance of a congestion control policy when tested outside of its training domain would be valuable for its adoption in practice.

The detection of changes in network conditions is an interesting challenge in itself, and could provide another direction for safe adoption of RL-trained policies. E.g., if we could detect when the network conditions greatly vary from the training conditions, we could fall back to some dedicated congestion control protocol. This problem is an instance of novelty detection (Schölkopf et al., 2000), and modern approaches employing deep networks could in principle be employed here (Zenati et al., 2018; Mandelbaum & Weinshall, 2017).

Another approach is continually adapting the policy to the observed network conditions. Recent approaches in meta-RL (Finn et al., 2017; Duan et al., 2016) could potentially be used to train policies that quickly adapt to network changes.

7. Related Work

Applications of RL to *highly specialized* contexts. Several past studies applied RL to highly specialized domains. Researchers proposed RL models for Asynchronous Transfer Mode (ATM) networks (Tarraf et al., 1995; Shaio et al., 2005); these predated deep RL and used shallow NNs. In addition, these studies are specially tailored to ATM (a proposed alternative to Internet protocols in general use today); for example, (Tarraf et al., 1995) requires use of the number of cells in ATM's multiplexer buffer to measure congestion. (Hwang et al., 2005) employs RL to improve congestion control for multimedia networks. This predated deep RL, and also used *cooperating routers in the core network*, where utilization is directly visible, to reach a global equilibrium; in contrast, we target the widely deployed case of transport layer congestion control where senders act *individually with limited information*.

Recently, two studies applied Q-learning (Watkins & Dayan, 1992) to specialized domains: (Li et al., 2016) optimizes TCP for memory-limited IoT devices, and (Silva et al., 2016) applies Q-learning to delay- and disruption-tolerant networks (DTNs), which apply to scenarios without ongoing end-to-end connectivity such as in interplanetary networks, which does not fit the common case of Internet transport.

(Ruffy, 2018) applied deep RL to congestion control in *datacenters* and provided an OpenAI Gym environment for benchmarking RL-based congestion control for this context. Unlike Internet congestion control where each sender acts independently, since a datacenter is administered by a single organization, (Ruffy, 2018) can leverage *global visibility of the network* to optimize a *network-wide reward function*.

Application of non-deep RL to Internet congestion control. (Kong et al., 2018) presents an RL solver based on the TCP framework. Unlike our work, (Kong et al., 2018) employs a rather antiquated RL algorithm (SARSA, (Sutton et al., 1998)), as opposed to deep RL. In addition, the action space and number of possible reward values are both extremely small (of size 4), greatly limiting the ability of the model to both learn and react to patterns. Lastly, the RL scheme is compared only to a variant of TCP from 1999 (namely, TCP NewReno (Floyd & Henderson, 1999)) and to the Q-learning scheme in (Li et al., 2016)).

Offline optimization of congestion control based on explicit assumptions about the network. Remy (Winstein & Balakrishnan, 2013) is an offline optimization framework for Internet congestion control. Remy takes as input *explicit* assumptions about the network, such as specified ranges of link bandwidths, number of competing connections, and more, and also an optimization objective. Then, Remy generates a stochastic model of the network and computes a congestion control protocol that approximates the optimum with respect to this model. Thus, unlike RL, Remy does not learn to react to patterns in traffic and network conditions, but relies on human-specified assumptions. When the prevalent traffic/network conditions deviate from Remy's input assumptions this might result in poor performance.

Congestion control via online-learning (multi-armed bandits). PCC (Dong et al., 2015; 2018) employs online learning techniques to guide congestion control. While this provides valuable *worst-case* guarantees (namely, no regret), unlike Aurora, it does not learn the prevailing network regularities and adapt rates accordingly. Thus, while PCC does provide robustness, it does not customize to the experienced network conditions, and our evaluation showed scenarios where our approach provides higher performance.

8. Conclusion and Future Work

We presented Aurora, a congestion control protocol powered by deep RL. Our evaluation results show that even with fairly limited training, Aurora is competitive with the state-of-the-art. We highlighted crucial challenges that should be addressed to simplify real-world adoption of such schemes. In future work, we plan to extend our simulation software to include realistic ancillary data and support for multi-agent decision making. To facilitate further research on deep-RL-guided congestion control, we open-sourced our code and a test suite based on the OpenAI Gym interface.

Acknowledgements

We thank Huawei for ongoing support of our research on congestion control. The fourth author is supported by the Israel Science Foundation (ISF).

References

Arun, V. and Balakrishnan, H. Copa: Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX{ Association}, 2018.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

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

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. Openai baselines. https://github.com/openai/baselines, 2017.

Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. Pcc: Re-architecting congestion control for consistent high performance. In *NSDI*, volume 1, pp. 2, 2015.

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. USENIX{ Association}, 2018.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Floyd, S. and Henderson, T. The newreno modification to tcp's fast recovery algorithm. Technical report, 1999.

Fontes, R., Afzal, S., Brito, S., Santos, M., and Esteve Rothenberg, C. Mininet-WiFi: emulating Software-Defined wireless networks. In *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*, Barcelona, Spain, November 2015.

Ha, S., Rhee, I., and Xu, L. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*, 2017.

Hwang, K.-S., Wu, C.-S., and Su, H.-K. Reinforcement learning cooperative congestion control for multimedia networks. In *Information Acquisition, 2005 IEEE International Conference on*, pp. 6–pp. IEEE, 2005.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Kong, Y., Zang, H., and Ma, X. Improving tcp congestion control with machine intelligence. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, NetAI'18, pp. 60–66, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5911-5. doi: 10.1145/3229543.3229550. URL http://doi.acm.org/10.1145/3229543.3229550.

Li, W., Zhou, F., Meleis, W., and Chowdhury, K. Learning-based and data-driven tcp design for memory-constrained iot. In *Distributed Computing in Sensor Systems (DCOSS), 2016 International Conference on*, pp. 199–205. IEEE, 2016.

Mandelbaum, A. and Weinshall, D. Distance-based confidence score for neural network classifiers. *arXiv preprint arXiv:1709.09844*, 2017.

Mania, H., Guy, A., and Recht, B. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1800–1809, 2018.

Narasimhan, K., Barzilay, R., and Jaakkola, T. Deep transfer in reinforcement learning by language grounding. *arXiv preprint arXiv:1708.00133*, 2017.

R.Evans, J.Jumper, J. L. T. C. A. A. A. H. S. K. D. K. D. A. De novo structure prediction with deep-learning based scoring. In *Thirteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstracts)*, pp. 11–12, 2018.

Ruffy, Fabian, P. M. B. I. Iroko: A framework to prototype reinforcement learning for data center traffic control. In *NeurIPS Symposium*, 2018.

Schapira, M. and Winstein, K. Congestion-control throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pp. 122–128. ACM, 2017.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. Support vector method for novelty detection. In *Advances in neural information processing systems*, pp. 582–588, 2000.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

Shaio, M.-C., Tan, S.-W., Hwang, K.-S., and Wu, C.-S. A reinforcement learning approach to congestion control of high-speed multimedia networks. *Cybernetics and Systems: An International Journal*, 36(2):181–202, 2005.

Silva, A. P., Obraczka, K., Burleigh, S., and Hirata, C. M. Smart congestion control for delay-and disruption tolerant networks. In *Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on*, pp. 1–9. IEEE, 2016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*. MIT press, 1998.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Tarraf, A. A., Habib, I. W., and Saadawi, T. N. Reinforcement learning-based neural network congestion controller for atm networks. In *Military Communications Conference, 1995. MILCOM'95, Conference Record, IEEE*, volume 2, pp. 668–672. IEEE, 1995.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84(1-2):51–80, 2011.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Winstein, K. and Balakrishnan, H. Tcp ex machina: Computer-generated congestion control. *SIGCOMM Comput. Commun. Rev.*, 43(4):123–134, August 2013. ISSN 0146-4833. doi: 10.1145/2534169.2486020. URL http://doi.acm.org/10.1145/2534169.2486020.

Winstein, K., Sivaraman, A., Balakrishnan, H., et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, volume 1, pp. 2–3, 2013.

Yan, F. Y., Ma, J., Hill, G., Raghavan, D., Wahby, R. S., Levis, P., and Winstein, K. Pantheon: the training ground for internet congestion-control research. *Measurement at http://pantheon. stanford. edu/result/1622*, 2018.

Zenati, H., Foo, C. S., Lecouat, B., Manek, G., and Chandrasekhar, V. R. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.