# A Pragmatic Learning-Based Approach Toward Mastering Congestion Control for the Internet

XXX, 17 pages


## ABSTRACT

These days, taking the revolutionary approach of using clean-slate learning-based designs to completely replace the classic congestion control schemes for the Internet is gaining popularity. However, we argue that current clean-slate learning-based techniques bring practical issues and concerns such as overhead, convergence issues, and low performance over unseen network conditions to the table. To address these issues, we take a pragmatic and evolutionary approach combining classic congestion control strategies and advanced deep reinforcement learning (DRL) techniques and introduce a novel hybrid congestion control for the Internet named Orca[1]. Through extensive experiments done over global testbeds on the Internet and various locally emulated network conditions, we demonstrate that Orca is adaptive and achieves consistent high performance in different network conditions, while it can significantly alleviate the issues and problems of its clean-slate learning-based counterparts.


## 1 INTRODUCTION

### 1.1 Setting the Context

More than three decades of active research on congestion control (CC) with the objective of improving the performance of TCP (in terms of achieving higher throughput, lower delay, and fairness) have provided us with a plethora of CC schemes (In Linux kernel alone for instance, there are more than 15 different CC algorithms). Every year, a new wave of technologies and improvements in the design of the packet-switched networks adds fuel to the fire and starts a new wave of CC designs. A recent example is Google's efforts to design a new CC algorithm (BBR) to gain high throughput for their inter-datacenter communications, because Cubic [20] (the default TCP scheme in most platforms) is not adequate in high BDP (bandwidth delay product) networks [11].

The need for new TCP designs and the fact that a plethora of them currently exists suggest that current CC algorithms are either *jack of all trades, master of none*, or *laser-focused solutions* that only can do a good job in certain networks with certain characteristics and break in other networks[2]. Classic CC designs usually rest on certain assumptions about the network and hard-wire predefined events to predefined actions. However, this main strategy comes with a cost: when those assumptions are not held, performance degrades dramatically. That being said, we still lack an *adaptive* TCP algorithm that can achieve consistent high performance in different environments without being manually tuned for each one.

### 1.2 Motivations

**Why learning-based congestion control?** Learning-based designs have great potential to **adapt** themselves to various conditions without the need of being tuned or manually engineered for every single one of them. So, an adaptive TCP can potentially avoid costly procedures of manually engineering/tuning TCP schemes to specific scenarios. For example, Google engineers have been working for a few years on adapting BBR to other environments such as cellular environments. In contrast, with a learning-based approach, the tuning and adaptation of TCP to new environments/conditions can become an automatic procedure done by machines in a few days or weeks (instead of years). So, there is a great opportunity to employ learning-based approaches for removing the hard-wired mapping of predefined signals/events to predefined control actions done in classic TCP algorithms for the last 3 decades.

**Why not Learning-based congestion control?** These days, using fully learning-based approaches is becoming popular in different communities including the network community. However, clean-slate learning-based approaches bring their own problems and costs to the table. To shed light on the problems of (current) clean-slate learning-based approaches in the context of congestion control, we perform an experiment. We emulate a network with a minimum RTT ($mRTT$) of $20ms$ where bottleneck link bandwidth changes every 10 seconds as shown in Fig. 1 and has enough buffer to absorb the traffic (= $1.5MB$). Note that the time scale of changes in the link capacity is 500× more than $mRTT$, enough for a CC scheme to converge. We use four recent clean-slate learning-based schemes named Indigo [62] (an imitation learning-based), Aurora [27] (a DRL-based), Vivace [14] (an on-line learning-based), and Remy [59] (an off-line optimization-based), TCP Cubic (currently default TCP algorithm in Linux, Windows, macOS, and Android), and Orca to send traffic over this network and report the sending rate of them throughout time in Fig. 1 (none of these schemes has seen this network condition during their training). Considering the results of

[1] Orcas are believed to be the most intelligent mammals (excluding humans). They have learned that teaming-up for the hunt is always beneficial!

[2] For a brief overview of related work, please see Appendix A

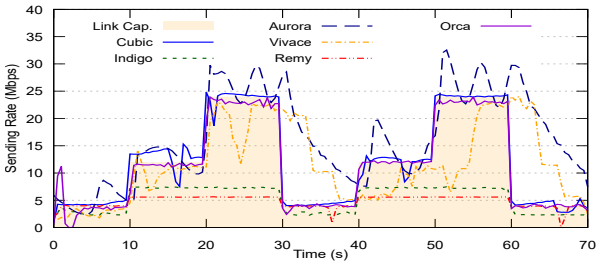



**Figure 1: Sending rate of clean-slate learning-based CC schemes, Cubic, & Orca over the Step-Scenario**

this experiment, we identify two key performance issues with current fully learning-based CC designs.

**1- Problem with unseen network scenarios:** Our first observation is that fully learning-based approaches experience serious performance issues over unseen conditions. These performance issues include serious under-utilization (e.g. Indigo and Remy during $[10s, 30s]$ in Fig. 1) and over-utilization (e.g. Aurora and Vivace during $[30s, 40s]$ in Fig. 1) of the network. In contrast, a classic CC scheme such as TCP Cubic is easy to reason about and achieves a somewhat predictable performance over unseen conditions, though its performance might not necessarily be very good.

**2- Convergence issue:** The second observation is that fully learning-based designs have convergence issues. We categorize convergence issues into two classes. In the first category, schemes settle down on the wrong equilibrium points (similar to Remy and Indigo in Fig. 1). In the second category, schemes either do not converge (similar to Aurora in Fig. 1) or may converge but very slowly (similar to Vivace in Fig. 1). We think that the convergence issues are caused by two factors: 1) the misleading assumption about the existence of *an* equilibrium point and 2) the lack of agile mechanisms to identify a better operating point.

**Overhead:** Besides, there is also another concern about the fully learning-based CC designs: Overhead. Generally, classic TCP schemes such as Cubic usually have very low computational overhead, while they employ a fine-grain time-scale for their control loops and react to every acknowledgment they receive. In contrast, clean-slate learning-based designs have potentially high computational overheads even when their control loops are longer than their classic counterparts (See Fig. 9). For instance, Aurora and Vivace have more than 100% CPU utilization when sending traffic over a $48Mbps$ link (detailed in section 5.3).

## 1.3 Key Design Decisions & Contributions

**A Pragmatic Learning-Based Approach:** The benefits of having a learning-based approach and the mentioned drawbacks and concerns of current clean-slate learning-based designs motivated us to propose a novel *pragmatic* approach for the design of a learning-based congestion control. We think that the practicality of a CC solution should not be sacrificed at the cost of the revolutionary aspect of it. To that end, our key design decision is to abandon the use of the fully clean-slate learning-based approach. Instead, we team-up with classic CC design and combine the classic approach with advanced DRL techniques to move toward mastering congestion control for the Internet. Our choice of DRL as the base learning technique is inspired by its enormous applicability to real-world problems (AlphaGo[52] and DeepMind's Atari [39] are among successful examples of applying DRL to practical scenarios).

**Why Reinforcement Learning? Why Deep?** Generally speaking there are three classes of learning algorithms: 1) supervised 2) unsupervised, and 3) reinforcement learning. Supervised and unsupervised learning are usually one-shot, myopic, and consider instant rewards, while reinforcement learning is sequential, far-sighted, and considers long-term accumulative rewards [32]. We see the CC problem as a sequential decision-making process that fits very well with the objectives of reinforcement learning. On the other hand, deep neural networks learn representations from raw inputs to recover higher-level features from lower-level ones. Therefore, in the context of CC, we utilize DRL because of its potentials and capabilities to learn from actual raw input data without relying on preprocessing or handcraft engineering for a task involving a sequential decision-making process.

**Contributions:** Our key contributions in this paper are:

- By demonstrating the problems of the current clean-slate learning-based CC designs, we revealed the need for a more practical learning-based design.
- We developed a novel distributed framework (available to community) to design and train new learning-based CC schemes. Using this framework, we proposed Orca, the first hybrid CC scheme that employs both advanced DRL techniques and classic CC strategies.
- We built, deployed, and successfully evaluated Orca over a global testbed on the Internet with servers located on five different continents. We showed that in contrast with other CC designs, it can achieve consistent high performance in different network conditions.

In what follows, we first overview Orca's design and its benefits. Later, in section 3, we dig deep into the design and describe the details of it. In section 4, we elaborate on the implementation of our framework and describe how Orca is trained. We explain our extensive experiments with Orca and report its results in section 5. We conclude the paper with a brief discussion about Orca.

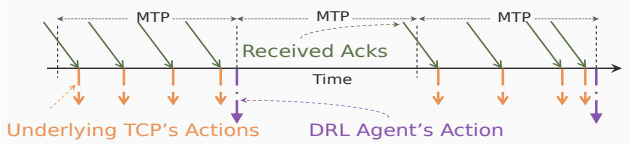



**Figure 2: Simplified time-diagram of Orca's two-Level control logic**

## 2 SYSTEM DESIGN OVERVIEW

### 2.1 Big Picture

Orca utilizes two levels of control: 1) fine-grain control and 2) coarse-grain control. In the fine-grain level, the underlying classic TCP algorithm performs its normal acknowledgment-based logic. Every received Ack packet triggers a predetermined adjustment for the current *cwnd* of the system. On top of this, in the coarse-grain level, every monitoring time period (MTP), the DRL-Agent monitors the environment's statistics (including both the statistics representing the network and the statistics representing the underlying classic TCP), calculates a new *cwnd* for the system, and enforces it. Later, the underlying classic TCP uses the enforced *cwnd* as its base *cwnd*, and continues its fine-grained adjustments to it. In practice, the reception rate of Ack packets is not necessarily constant. For instance, receivers might occasionally wait to accumulate Acks, or the available uplink bandwidth might change due to the possible coexistence of Ack packets and data packets of other users. Therefore, there might be MTPs without any incoming Ack packets. In these scenarios, DRL-Agent (similar to underlying TCP) does not perform any action at the end of MTP[3] (similar to 2nd MTP in Fig. 2). Finally, to smooth out the changes of *cwnd* through time and to limit excessive burstiness of traffic, Orca paces the outgoing packets using the calculated *cwnd* of the system at different times.

### 2.2 Benefits of Two-Level Control

**Continuous Probing and Convergence:** By letting the underlying TCP adjust *cwnd* during each MTP, Orca deliberately adds *disturbances* to the DRL-Agent's output. These disturbances reduce the chances of Orca settling down on the wrong equilibrium points. In other words, the underlying TCP's continuous adjustments of *cwnd* works as a *backup* probing mechanism and lowers the chances of system being trapped in wrong sending rates. This (as Fig. 1 illustrates) improves the convergence property of Orca. In section 5.2, we elaborate more on the impact of these backup probing mechanisms.

[3]As usual, the existing timers still may trigger actions of the underlying TCP when no Ack packet is received

**A More Predictable Behavior Compared to Clean-Slate Designs:** Classic TCP is easy to understand and somewhat predictable, though it is not adaptive to various network conditions and experiences performance issues. On the other hand, learning-based CC designs are usually hard to reason about, though they can potentially learn to adapt to different network environments. By combining these two approaches and having a more pragmatic two-level control architecture, Orca attempts to leverage the best of both worlds. We elaborate more on achieving consistently high performance in different network environments by following this two-level control structure in section 5.

**Overhead:** Another benefit of Orca's two-level control mechanism is to achieve a better trade-off curve for performance and overhead compared to fully learning-based approaches (Fig. 9 and Fig. 10). We elaborate more on this property in sections 5.3 and 5.4.

**A More Efficient and Faster Training:** Fine-grain control done by underlying TCP significantly reduces the action space of DRL-Agent and helps the system to learn a more general model faster during the training phase (See Fig. 4). We elaborate more on this benefit in section 4.3.

## 3 ORCA'S DESIGN

Orca's main building blocks, as shown in Fig. 3, includes Monitoring Block, Reward Block, DRL-Agent, and the underlying TCP scheme. In what follows, we describe these components and their responsibilities in detail.

### 3.1 Monitoring Block

The monitoring block works as the eyes of the system. It is responsible for gathering information from the environment by reading packet statistics from the lower layers and feeding the DRL-Agent with the updated input representing the environment. In particular, the monitoring block works as a shim layer and continuously generates the required packets' statistics by observing the incoming Ack packets and various timers. Having the monitoring block implemented as a shim layer in the Kernel enables us to make the process of gathering required statistics fast and independent of the underlying TCP scheme. The features that the monitoring block gathers are collectible statistics that can be measured/monitored on the fly from the network. We consider the statistics shown in Table1. The calculation of these statistics occurs from the last report time to current report time, except the last four ones: $sRTT$, $cwnd_u$, $thr_{max}$, and $d_{min}$.

**Normalization**: Instead of feeding the exact values of gathered statistics to the agent, we use normalized statistics. This helps the agent generalize the network environments that it observes during the training sessions to unseen environments and to achieve a better model. On the other hand,

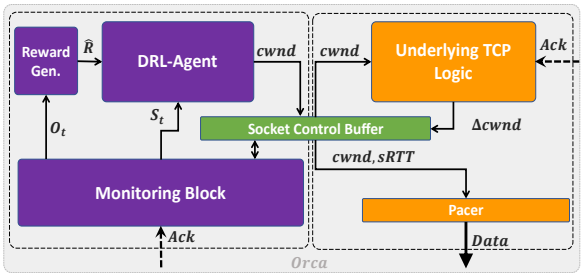



**Figure 3: High-level block diagram of Orca**

normalization helps the agent treat input signals in the same way and avoid exaggerating the impact of an input statistic with very large values on the final model compared to the one with small values.

**Recurrent structure**: The foundation of solving a general RL problem is to model the task as a Markov Decision Process (MDP) [55]. However, different network environments do not necessarily have a Markov property. Therefore, to address this issue, we use a recurrent structure [23, 47]. Since at no time does the agent have the direct knowledge of the current exact network statistics (including delay and bandwidth), the agent can only infer it as a latent variable from the observed features. Therefore, we aggregate partial information from the past and capture the dependency in the sequential data. To that end, the $m$ history of feature vectors is concatenated to the current observed features to generate the final state. So, the final state (i.e., the input to the DRL-agent block) at time $t$ becomes the vector $s_t = (o_t, o_{t-1}, ..., o_{t-m})$, where $o_t$ and $o_{t-i}$ represent the statistics that are observed at time $t$ and the statistics observed in the previous $i$th report, respectively. Using this state representation, we can apply the MDP style formulation.

## 3.2 Reward

One of the important factors impacting the performance of a DRL system is the choice of reward/objective function. Intuitively, the reward that the agent gains at each step quantifies its performance and provides it with the criterion to improve its sequence of actions. To choose the reward function, we start from an important and well-studied metric in the context of congestion control (and flow control) called Power. Power defined as $Power = \frac{Throughput}{Delay}$ was introduced by Giessler *et al.* [19] in 1978 [4]. Later, Gail and Kleinrock showed that the operation point where Power is maximized is the optimal point both for the network and for the individual flow [18, 30].

[4]Throughout this paper we use delivery rate and throughput of the system interchangeably

**Table 1: Statistics generated by Monitoring block**

| | |
|---|---|
| $thr$ | The average delivery rate (throughput) |
| $l$ | The average loss rate of packets |
| $d$ | The average delay of packets |
| $n$ | The number of valid acknowledgment packets |
| $m$ | The time between the last report and current report |
| $sRTT$ | The smooth RTT of packets so far |
| $cwnd_u$ | The current congestion window |
| $thr_{max}$ | The maximum value of delivery rate so far |
| $d_{min}$ | The minimum value of packet delay so far |

Maximizing Power reflects the objective of maximizing throughput while minimizing the delay in the network. On top of that, we also directly consider the objective of minimizing the loss rate of packets for the agent. Therefore, putting all together, and using the well-studied metric of Power as the base for the reward function, Orca employs the following normalized reward, $\hat{R}$, that motivates the agent toward maximizing the throughput, while minimizing loss rate and delay:

$$\hat{R} = R/R_{max} = \left(\frac{throughput - \zeta \times loss}{delay}\right)/\left(\frac{thr_{max}}{\beta \times d_{min}}\right) \quad (1)$$

In Eq. 1, $\zeta$ is a coefficient determining the relative impact of the loss rate compared to the throughput rate.

Measuring the maximum Power value in practice faces an interesting paradox. Intuitively, a *distributed* scheme needs to probe/measure both minimum delay and maximum throughput of the network to detect the maximum value of Power. However, maximum throughput is only observable by sending at rates more than the network's bandwidth while observing minimum delay of the network requires sending at rates less than the network's bandwidth. In other words, maximum throughput and minimum delay cannot be observed/measured at the same time. In more technical terms, Jaff proved that "Power is nondecentralized" [26]. This phenomenon impacts the agent's interpretation of the environment and the actual maximum reward that it can achieve, especially in dynamic network environments [5]. To address this issue, we allowed the agent to make small queuing delay to have extra room for observing the maximum bandwidth. That is why we add coefficient $\beta$ ($> 1$) in Eq. 1. In other words, when delay ($d$) in the network satisfies $d_{min} \leq d \leq \beta \times d_{min}$, the agent can still get the maximum reward.

## 3.3 DRL-Agent

The DRL-Agent is the heart of the system. It takes proper actions toward maximizing its achievable rewards based on the environment's statistics generated by the monitoring block.

**Huge Action Space and the Contradiction with the Real-Time Nature of the Problem:** The ultimate desired

[5]This dynamism can include variable traffic patterns, variable number of users, intrinsic variations of the link capacities, etc.

outputs of any congestion control algorithm are the congestion window (*cwnd*) and the pacing rate ($p_{rate}$) through time. However, the space for exploring these values is huge. On the other hand, the problem of controlling congestion in the network has a real-time nature. Both of these issues are unique. For instance, in a game environment such as Go (as in [52]), Atari games (as in [39]), chess (as in [53]), etc, the action space at each step is a limited set of movements (e.g. pushing an up or down button on the console's controller) and the time for taking an action is not necessarily bounded, while in our context, there is no limited set of *cwnd* values and actions should be taken promptly. The bottom line is that simply letting the DRL-agent explore the immense action space to find the best action values in a timely manner is not feasible in our context.

Therefore, to deal with the real-time nature of the CC problem and exploration of the huge action space, we first need to reduce the action space in a wise manner. To that end, Orca uses the underlying TCP's *cwnd* ($cwnd_u$) as a base value for exploring a better *cwnd*. In other words, instead of directly determining *cwnd* as the output, DRL-Agent determines a parameter called $\alpha$ as the output which relates the final *cwnd* to $cwnd_u$ by Eq. 2.

$$cwnd = 2^{\alpha} \times cwnd_u \qquad (-2 < \alpha < 2) \tag{2}$$

Exploiting the estimations made by the underlying TCP, greatly simplifies the exploration phase and consequently lowers the convergence time (see section 4.3), while *cwnd* can still be increased (decreased) exponentially when the agent chooses $\alpha \geq 1$ ($\alpha \leq -1$) consecutively. We let the agent have extra space for $\alpha$ (i.e. $-2 < \alpha \leq -1$ and $2 > \alpha \geq 1$) to explore the increase/decrease of *cwnd* with more freedom.

After having *cwnd*, $p_{rate}$ is calculated using Eq. 3.

$$p_{rate} = \frac{cwnd}{sRTT} \tag{3}$$

## 3.4 Learning Algorithm

**Reinforcement Learning Formulation:** We formulate the CC problem in a reinforcement learning setting. In reinforcement learning paradigm, an agent interacts sequentially with the environment with the goal of learning reward-maximizing behavior. At each time step $t$, agent observes state $s_t \in \mathcal{S}$ and selects action $a_t \in \mathcal{A}$ with respect to its policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, receiving a reward $r_t = r(s_t, a_t)$ (as defined in Eq. 1) and a new state $s_{t+1}$. The return is defined as discounted sum of rewards $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$. The ultimate objective is to find the optimal policy $\pi_\theta$, with parameters $\theta$, which maximizes the expected return $J(\theta) = \mathbb{E}[R_0]$.

**Agent's Algorithm:** Orca's agent uses actor-critic reinforcement learning to learn the policy (known as actor) and action-value function (known as critic) concurrently where both functions are parameterized by deep neural networks. The action-value function, which is defined as $Q_\pi(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$, measures the expected return when performing action $a_t$ in state $s_t$ following policy $\pi$.

**Algorithm 1** Orca's Learning Algorithm

**Input:** Replay Memory $D$, Learning rate $\eta$ and $\rho$, Batch size $N$

1: Randomly initialize network weights ($\theta$, $w_1$, $w_2$)
2: Initialize target networks weights ($\theta'$, $w_1'$, $w_2'$)
3: **for** t=1 to T **do**
4: Sample $N$ transitions ($s$, $a$, $r$, $s'$) from $D$
5: Compute the target function:

$$y = r + \min_{i=1,2} Q_{w_i'}(s', \pi_{\theta'}(s') + \epsilon)$$

6: Compute the gradient for actors and critics:

$$\nabla_{w_i} L(w_i) = \frac{1}{N} \sum \nabla_{w_i} (Q_{w_i}(s, a) - y)^2$$

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$$

7: Update the parameters of actor and critics:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta),\ w_i = w_i - \rho \nabla_{w_i} L(w_i)$$

8: Update the parameters of target networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta',\ w_i' \leftarrow \tau w_i + (1-\tau) w_i'$$

9: **end for**

Orca's actor takes state $s$ and outputs deterministic action $a = \pi_\theta(s)$. Since Orca operates in a continuous action space, its agent is built on top of the twin delayed deep deterministic policy gradient algorithm (TD3) [17], a variant of deep deterministic policy gradient algorithm [34], which is designed to learn continuous action efficiently.

Consider $J(\theta)$ the expected return of the agent, where $\pi_\theta(s)$ is a parameterized policy ($J(\theta) = \mathbb{E}[Q_{\pi_\theta}(s, \pi_\theta(s))]$). By applying the deterministic policy gradient theorem [54], the gradient of the expected return can be written as:

$$\nabla_\theta J \approx \mathbb{E}[\nabla_a Q_{\pi_\theta}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)] \tag{4}$$

By denoting the learning rate as $\eta$, the policy update rule is as follows:

$$\theta \leftarrow \theta + \eta \nabla_\theta J \tag{5}$$

To compute the gradient in Eq. 4, a true action-value function $Q_{\pi_\theta}$ used to evalute the policy $\pi_\theta$ is required. $Q_{\pi_\theta}(s, a)$ can be approximated by a parameterized critic $Q_w(s, a)$. The $Q_w$ parameters can be trained by minimizing the mean squared Bellman residual [7]:

$$L(w) = \mathbb{E}[(Q_w(s, a) - (T_{\pi_{\theta'}} Q_{w'})(s, a))^2] \tag{6}$$

where $T$ is the Bellman operator:

$$(T_\pi Q)(s, a) = r(s, a) + \gamma\, \mathbb{E}[Q(s', \pi(s'))] \tag{7}$$

The main idea of minimizing critic loss in Eq. 6 is to make the $Q_w$ close to a target function that uses a separate set of policy and Q function parameters (target networks $\theta'$, $w'$) to stabilize the training [40]. A good policy relies on a good estimation of critic $Q_w(s, a)$ and a reduction of estimation error directly improves the policy. As was done in TD3, we utilize

a variant double Q-learning to address the overestimation issue of critic. The overall Orca's algorithm is summarized in Algorithm 1. Note that the critic and target networks are only used during training (not needed for evaluation).

### 3.5 An Efficient Distributed Learning

Orca's ultimate goal is to learn one policy that can achieve high performance in diverse environments and network conditions with a single set of parameters. However, the optimal policy in different network environments with different dynamics varies. For instance, a CC scheme that can work in a small BDP network might not necessarily work well in a large BDP environment. This leads to practical challenges for training the agent over multiple environments.

**1- Catastrophic Forgetting:** One of the challenges in the training of deep neural networks (DNN) is catastrophic forgetting [16, 36, 37, 49]. Catastrophic forgetting is due to the tendency for knowledge of previously learned environments (e.g., environment A) to be abruptly lost as information relevant to the new environment (e.g., environment B) is incorporated. This phenomenon occurs specifically when the neural network is trained sequentially on multiple environments because the weights in DNN that are important for environment A are changed to meet the objective in environment B [29].

**2- Handling the Extended Training Time:** Another important challenge for training a single agent on many environments is extended training time. During training, an agent needs to interact with the environment, gather many experiences, and explore possible policies. In contrast with a game environment where sampling from the environment might not be restricted by physical factors such as time, in the context of a real-time problem such as congestion control, gathering experiences from the environment requires the passage of time. For a single agent to learn through a wide range of network scenarios, the training time would become too long and impractical.

**Scaling Out the Learning:** To address these challenges and learn effectively/efficiently in rich and dynamic environments, we scale out Orca's learning algorithm. To that end, inspired by prior work such as Gorilla [42] and A-Pex [24], we utilize a distributed architecture. We modify TD3 and combine it with our distributed framework. The architecture consists of 3 main components: Actor, Replay Memory, and Learner.

**Actors:** An actor is responsible for generating new experiences corresponding to an environment. Each actor in our framework has a replica of the learner's policy network $\pi_\theta$ which is periodically synchronized with the latest model in learner. By observing new state $s$, an actor selects action $a$ following the policy, with Gaussian noise added to it to encourage exploration. By executing $a$ in the corresponding environment, actor observes new reward $r$ and new state $s'$. Then, it stores the experience tuple $e = (s, a, r, s')$ in its local experience buffer. New experiences in local buffer will be periodically added to a global replay memory.

**Replay Memory:** A replay memory $D$ is a global memory, implemented either in a distributed or centralized way, that gathers the experience tuples generated by actors. However, its capacity needs to be large enough to store the history of actors' experience tuples. Orca does not distinguish experience tuples of different actors and keeps a balanced number of experience tuples from each environment to let the agent learn the policy which is environment agnostic.

**Learner:** A learner maintains and updates the neural network models, including policy, critic, and target networks. For each update iteration, the learner randomly samples a mini-batch of experience tuple $e = (s, a, r, s')$ from the replay memory, computes the gradients from sampled data, and updates the corresponding parameters following algorithm described in Algorithm 1.

By decomposing the acting phase and the learning phase, actors can execute over their own environments asynchronously and the learner can update the model without being blocked by actors. This framework enables us to exploit the parallelism, learn a model that can solve various environments, and help the framework achieve a good generalization.

## 4 IMPLEMENTATION AND TRAINING

### 4.1 Orca

Orca only requires modifications on the sever-side and it works smoothly when any other TCP scheme is used on the client-side. As a prototype version, we implemented DRL-Agent in user-space on top of Tensorflow [1], while the monitoring block and rest of the design were implemented in Kernel (on top of Linux Kernel 4.13). We used new socket options for communication among Kernel and user-space blocks. We chose TCP Cubic as Orca's underlying TCP due to its popularity in today's platforms (it is default TCP scheme in Windows, macOS, Linux, Android, etc.). For brevity, we omit the details of the implementation.

### 4.2 Distributed Actors-Centralized Learner

Although the framework is capable of being used to train directly in the wild, here, our approach is to do the training with controlled emulated environments, because it greatly simplifies the management and cost of the training sessions (in terms of renting various servers around the globe to represent various network environments on the Internet). To that end, we use a network emulator (Mahimahi [43]) to emulate different network environments. Mahimahi sends real packets over TUN/TAP interfaces in Linux. Using that, we are

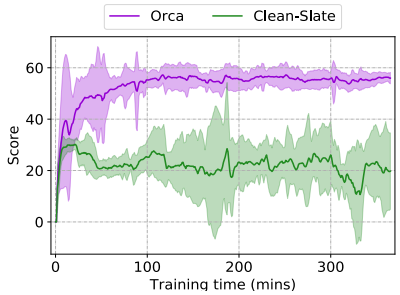



**Figure 4: Training curves of Orca and its Clean-Slate counterpart (averaged over all sessions)**

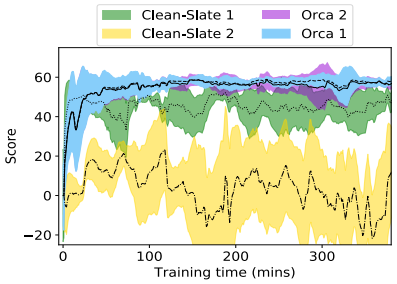



**Figure 5: Training curves of 2 sets of actors for both Orca and its Clean-Slate counterpart.**

| | GENI Servers |
|---|---|
| AWS Servers | California (Stanford) |
| Australia (Sydney) | California (UCLA) |
| Asia-South (Mumbai) | New York (Cornell) |
| Asia-North East (Seul) | Texas (UTDallas) |
| Asia-Middle East (Bahrain) | Tennessee (UTC) |
| South America (Sãu Paulo) | Kentucky (UK) |
| Europe-Central (Frankfurt) | Colorado (UCB) |
| Europe-North (Stockholm) | Indiana (IU) |
| Europe-West (Paris) | Washington (UW) |
| | Wisconsin (UWM) |

**Figure 6: Location of AWS/GENI servers used in real-world experiments**

able to directly employ the Linux stack and real-world client-server applications for sending traffic, instead of using network simulators with simplified network stack codes. That lets the agent have an experience close to the ones that she will have in real-world networks. We characterize different network environments with 3 key parameters: 1) bottleneck link bandwidth 2) minimum RTT (delay) 3) bottleneck link's buffer size. Employing our distributed framework, we distribute different actors through different physical servers in a cluster. Each actor observes a certain network environment and interacts with that environment. Actors are in contact with the centralized learner located on a separate physical server. To avoid the communication delays between learner and replay memory units and make learning faster, we use a centralized approach to implement replay memory and put replay memory and the learner on the same physical server.

## 4.3 Training

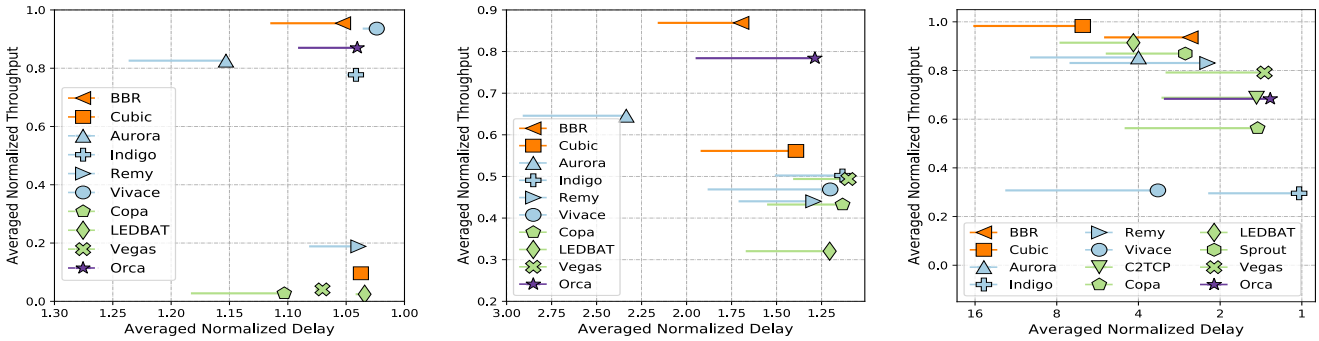
We use a pool of servers combined having 320 CPU cores as our pool of actors. These servers are connected to edge-switches with high-speed links. An aggregate-switch connects edge-switches to the learner. Orca's learner runs on a Dell PowerEdge R730 server with 48 CPU cores, 256GB RAM, and equipped with GEFORCE RTX 2080 TI GPU. Totally, we use more than 256 actors interacting with different environments during training (their characteristics are shown in Table 2). To balance the impact of each environment on the final model, we collect certain number of samples from each worker. When a worker provides the learner with the total requested samples, it has done its job. When all actors are done, a training round ends and we start a new round of training. One session of training includes multiple training rounds. To make sure that the gained model at the end of a training session is not impacted by the randomly initialized weights, training sessions are repeated 5 times (Table 3 in Appendix C shows the parameters used during the training.)

**Table 2: Range of emulated env. during the training**

| B.L Bandwidth | Minimum RTT | B.L Buffer Size |
|---|---|---|
| $[6Mbps - 192Mbps]$ | $[4ms - 400ms]$ | $[3KB - 96MB]$ |

In addition to training Orca, we use the same setting to train a purely DRL-based version that we call Clean-Slate scheme. The Clean-Slate version is achieved by not using any underlying TCP as a fine-grain control. We record the rewards that actors gain through time for both schemes during the training sessions. To summarize the results of all training sessions, we average rewards of all actors (called score) for each scheme over all 5 training sessions. The moving average of scores for each scheme through time and the standard deviation of them is shown in Fig. 4.

**Main Takeaway:** Comparison between the performance of Clean-Slate version and Orca demonstrates the benefits of Orca's pragmatic two-level control architecture even during the training. Fine-grain control by underlying TCP speeds up the training session dramatically compared to Clean-Slate version. In other words, Fig. 4 illustrates that during the same length of training, Orca achieves 2× more score compared to Clean-Slate version (which can be interpreted as learning 2× more environments). To have a better picture, we randomly choose one of the training sessions and sort actors based on their average gained rewards in the Clean-Slate scheme. Using that, we find the top and the bottom 20% of the actors and report the average score of them for both schemes during training (Fig. 5). Clean-Slate version is able to achieve good performance for the top 20% of actors (Clean-Slate#1 in Fig. 5) while its performance remains bad for the bottom 20% of actors (Clean-Slate#2 in Fig. 5). However, fine-grain control in Orca helps it to boost the performance of both sets of actors and steer the system toward achieving a better general model which learned more environments faster.




**(a) Inter-Continental experiments** **(b) Intra-Continental experiments** **(c) Highly Varibale Link experiments**

**Figure 7: Normalized Avg. delay (icons), 95%tile delay (end of lines), and avg. throughput in diff. experiments**

## 5 EVALUATION

Now, we demonstrate advantages of Orca's pragmatic design approach in real-world scenarios over various state-of-the-art clean-slate learning-based CC designs including Aurora [27], Indigo [62], Vivace [14], and Remy [59][6] and different TCP schemes including BBR [11], Cubic [20], LEDBAT [51], and Vegas [9]. In particular, we show Orca's **consistent high performance** in different networks (section 5.1), its **deployment friendliness** including its **low overhead** (sections 5.3 and 5.4) and **TCP friendliness** (section 5.6), and Orca's **fairness** property (section 5.5). Also, we show that Orca's high performance is **insensitive** to the different AQM (active queue management) designs that might be used in the network (section 5.7). In addition, to have a better understanding of our system, we look under the hood of Orca (section 5.2) and discuss its dynamics[7].

### 5.1 Consistent High-Performance

Here, we demonstrate that Orca achieves consistent high performance over different unseen network environments on the Internet by using the model trained in section 4.3 over simplified emulated environments. To that end, we make a global testbed using our servers located at an anonymous location (just for now, to keep the submission anonymous!), AWS servers located around the globe, and GENI [8] servers located around the USA. Location of these servers are shown in Fig. 6. Using these servers we can cover a variety of network conditions including various delays and bandwidths.

**Inter-Continental Scenarios:** Inter-continental network environments are challenging and complicated environments due to their large intrinsic BDPs, their multi-hop and multi-entity architectures, the use of different queue management and traffic shaping schemes unknown to the end-points, etc. To do the experiments over these challenging networks, we send a flow using the scheme under the test for 60 seconds over each client-server connection. We test all schemes back-to-back. Then, we repeat the tests 5 times. To measure the delay of packets, we synchronize client-server pairs to NTP servers before sending each flow to the network. To report the overall performances of all schemes under the test, for every scenario (specific client-server connection), we normalize delay and throughput performances of all schemes to the minimum delay and maximum throughput achieved on that scenario, respectively. Then, we average all normalized values over all scenarios. We use 3 performance metrics: average One-way delay, 95%tile delay, and average throughput. The results are shown in Fig. 7a.

**Intra-Continental Scenarios:** Intra-continental network environments are another important part of the global Internet with their small to medium BDPs and low latency properties. Here, we use the same experiment setup that we employed for inter-continental experiments and report the average normalized values. The results are shown in Fig. 7b.

**Highly Variable Link Scenarios:** In addition, to test the behavior of different schemes over highly variable networks, we use cellular network environments. Cellular networks are known to be very challenging networks due to their high access link bandwidth fluctuations. In particular, we use traces from AT&T and T-Mobile cellular networks ([60]) and send traffic over these cellular traces using Mahimahi [43]. We repeat the tests 5 times and report the averaged normalized values of performance metrics. In addition to the schemes that we compared so far, we add two more schemes namely Sprout [60] and C2TCP [2] that are engineered for cellular networks. The results are shown in Fig. 7c. For all three scenarios combined, we gathered more than 52GB of data.

**Main Takeaway:** The right top region of the graphs in Fig. 7 is the desired operation region for any CC algorithm:

[6]For Indigo and Aurora, we use the same model trained by their authors over variety of network scenarios. For Remy, similar to others (e.g. [5, 27, 62]) we use the model trained for a 100× range of link rates in prior work [5].
[7]For our in-field tests, we used some of the tools provided by prior work [62]

gaining high throughput while achieving low delay. As expected, some CC schemes do a good job in a certain scenario, while they operate poorly in the other ones. That is one of the key disadvantages of CC algorithms proposed in the last 3 decades. In contrast, Orca's pragmatic learning-based approach yields consistent high performance in all experiments. For a more detailed version of the results, please see Appendix B.

**More Observations: #1-** Although Orca in part exploits TCP Cubic's fine-grain control logic, its performance is way beyond the performance of Cubic in different scenarios. That illustrates the important role of coarse-grain control by DRL-Agent in Orca's pragmatic design.

**More Observations: #2-** Performance comparison between Orca and clean-slate learning-based schemes (colored blue in Fig. 7) reveals the important aspect of Orca's two-level control. This architecture not only makes Orca a more practical solution (in terms of overhead, etc.), but also leads to much higher performance compared to them on average.

**More Observations: #3-** Excluding Orca, among other schemes, BBR achieves better consistent performance in these experiments. However, as we show in section 5.6, BBR's performance comes with the cost of being very aggressive toward other flows in the network. In contrast, Orca achieves its high performance without being unfriendly to other TCP flows (see Fig. 13 and Fig. 14).

**More Observations: #4-** Delay-based designs (colored green in Fig. 7) show serious throughput performance degradation in inter-continental and intra-continental scenarios, while they perform good in highly-variable link emulations. We think one of the key reasons is coexistence with throughput hungry TCP flows on the Internet (in cellular emulations, network is dedicated to one scheme at a time).

**More Observations: #5-** The closest learning-based design to Orca in terms of using the DRL approach is Aurora. However, we find out that Aurora learned to be very aggressive and often sends as many packets as it can to the network. We verify this unhealthy behavior of Aurora in section 5.5.

## 5.2 Under The Hood

It is known that we still lack a theoretical understanding of many methods that are currently used in learning-based approaches, particularly in deep learning (for instance, check out Rahimi's Test of Time award speech at NIPS [48] and Lencun's response to it [31]). That means the question of why learning-based methods work very well in practice is still an open question seeking solid theoretical answers. That being said, here, we intuitively attempt to shed light on the question of "How Orca achieves adaptiveness?". To that end, we describe Orca's behavior through using two scenarios shown in Fig. 8: 1) when link capacity suddenly halves (representing

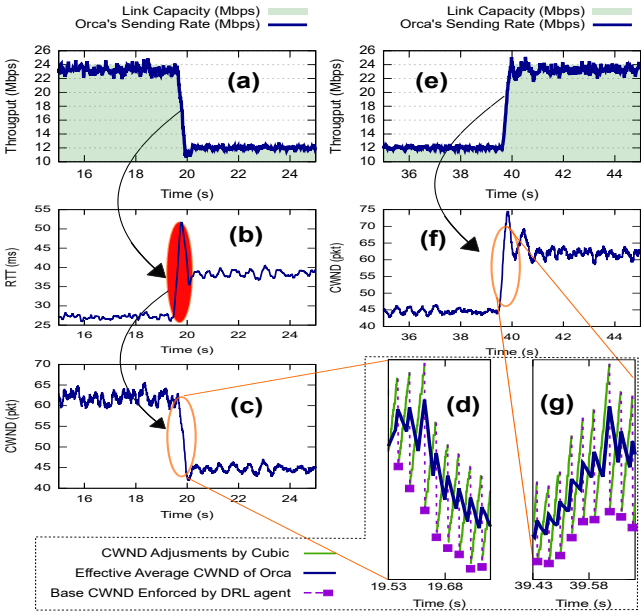



**Figure 8: Details of the Orca's actions in two scenarios**

decrease in available BW) 2) when link capacity suddenly doubles (representing increase in available BW). Note that DRL-Agent receives more than 70 different input signals (including the history of collected statistics) during each MTP. Following all these signals and reverse-engineering the model learned by Orca is impractical. That's why the following explanations are only our hypothesis.

In the first scenario (left graphs in Fig. 8), when link capacity halves, Cubic, as a loss-based CC, still attempts to increase *cwnd* (green ramp-up lines in Fig. 8.d), because it has not seen packet loss yet. This normally causes bufferbloat issue of Cubic. However, in Orca, DRL-Agent detects a sudden jump in the delay of packets (Fig. 8.b), while there is no gain in its throughput. For DRL-Agent, this means getting lower rewards. So, to increase its future accumulated rewards, DRL-Agent enforces the decrease in *cwnd* by setting a new base *cwnd* (violet squares in Fig. 8.d) at the end of each MTP. Enforcing the decrease in *cwnd* by DRL-Agent continues up to a point where it will not lead to gaining more rewards. After that, DRL-Agent stops decreasing base *cwnd* of the system and Orca operates on a new equilibrium point.

The second scenario demonstrates the role of underlying TCP's fine-grain control which works as a backup probing mechanism. When link capacity increases, the classic probing mechanism of Cubic helps the system detect higher available throughput (Fig. 8.g). As a result, DRL-Agent measuring higher reward increases the base *cwnd* of the system (violet squares in Fig. 8.g). DRL-Agent has learned to do this increase conservatively to control the bufferbloat issue of

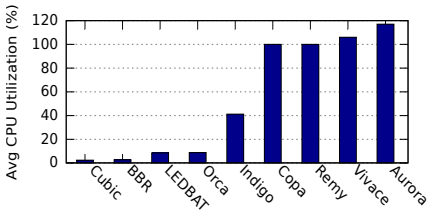



**Figure 9: Overhead of DRL agent compared with other schemes**

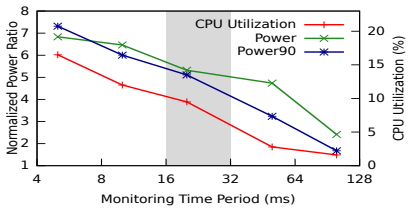



**Figure 10: Performance & overhead trade-off across diff. MTPs**

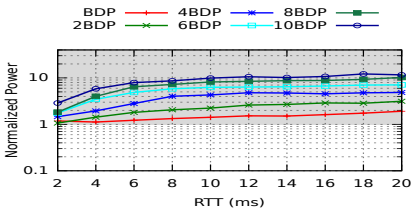



**Figure 11: Orca's Perf. (Norm. to Cubic) across RTTs and buffer sizes**

Cubic (note that DRL-Agent enforces a lower *cwnd* than the one calculated by Cubic). Then, Cubic starts from a larger *cwnd* as the base (violet squares in Fig. 8.g) and continues probing more bandwidth by increasing *cwnd* (green ramp-up lines in Fig. 8.g). This cycle of increasing *cwnd* by Cubic, observing more reward at next MTP by DRL-Agent, and enforcing a larger base *cwnd* by DRL-Agent continues until the link is fully utilized. After that, DRL-Agent won't see any increase in the throughput (and its reward). Therefore, it stops the loop by stopping further increase of the base *cwnd*. So, any unhelpful further temporary increase in *cwnd* by Cubic during an MTP, is revised by DRL-Agent at the end of that MTP and a new equilibrium point is achieved. Since Orca smooths out the outgoing traffic, Orca's effective *cwnd* becomes the average *cwnd* (thick blue lines in Fig. 8).

## 5.3 Overhead

To investigate DRL-Agent's system overhead and compare it with other schemes, we use various state-of-the-art CC schemes and send traffic from a server to a client over an emulated network (with 48Mbps bottleneck link and 20ms RTT) for 120 seconds and measure the average CPU utilization of these schemes on the sender side. For BBR and Cubic which are fully implemented in the kernel, we report the CPU utilization of iperf (used for sending their traffic). To have a fair comparison and reduce the impact of initialization phases required by some of these schemes, we exclude the first few seconds of the experiment for all schemes. Results are shown in Fig. 9. As we expected, DRL-Agent achieves very low overhead compared to other learning-based and even non-learning-based CC schemes such as Copa. Although the current user-space implementation of DRL-Agent has low overhead, we believe that Orca's final optimized kernel version will achieve even lower overhead.

## 5.4 Monitoring Time Period

Here, we investigate the impact of MTP value on the performance and overhead of the system. Intuitively, MTP determines the frequency of Orca's coarse-grain control. To that end, we use two sets of experiments. In the first set, we use the same network setup described in section 5.3, change the values of MTP, and record the impact of it on CPU utilization and performance of the system. To report the relative performance of the system, we use Power ($\frac{\text{throughput}}{\text{average delay}}$) and Power90 ($\frac{\text{throughput}}{\text{90\%tile delay}}$) that a pure Cubic flow achieves as the base and normalize the Power and Power90 achieved by Orca to those values. Fig. 10 depicts the results. There are two takeaways here. First, even with large MTP values, Orca's performance still is better than a pure TCP Cubic. Second, for smaller values of MTP, Orca's performance increases at the cost of an increase in its overhead. [8]

In the second set, we fix MTP to 20*ms* and change RTT and buffer size of the network to see their impact on the performance of Orca, particularly when RTT is less than MTP. We normalize the Power in each scenario to the Power gained by a pure Cubic flow on the same scenario. Results are reported in Fig. 11. Even when RTT of the network compared to MTP is relatively very small (10× less), Orca on average achieves 2× higher performance compared to a pure Cubic flow. Also, for large buffer sizes, Cubic makes bufferbloat and its performance degrades, while Orca keeps preforming well. That is why relative performance of Orca to Cubic increases in Fig. 11 for larger buffer sizes. That shows the role of a coarse-grain control in Orca's performance.

The bottom line is that MTP provides a degree of freedom to trade-off among performance and overhead. To have a balanced performance and overhead and by considering the normal a few tens of milliseconds delay in the Internet, we fixed $MTP = 20ms$ throughput this paper, though one can imagine that MTP value can be dynamically tuned based on the network environment to gain even higher performances. We leave those potentially sophisticated alternative approaches to future work (e.g. Orca's DRL-Agent might be used to learn and determine MTP).

## 5.5 Dynamics of Orca

How Orca behaves in the presence of other Orca flows? To answer that and compare its dynamics with other CC

[8] The overhead reported here are from the user-space implementations of Agent. Potentially, a kernel version will have much lower overhead.

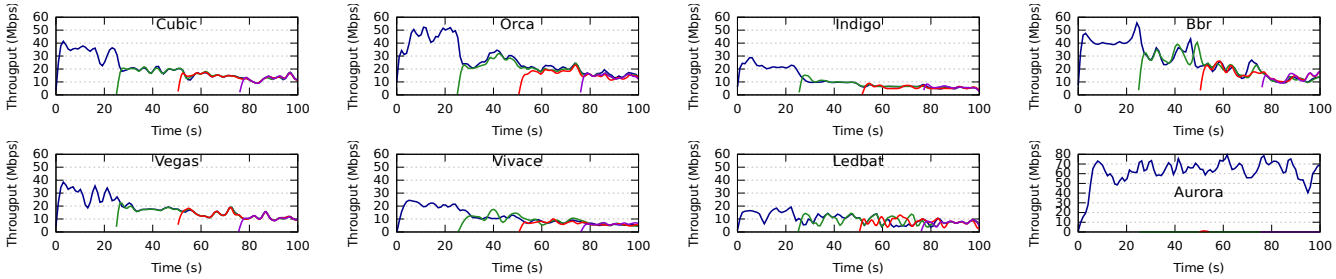



**Figure 12: Throughput dynamics of different flows competing on an Internet link for various congestion controls**

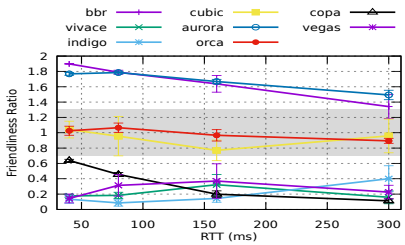



**Figure 13: Friendliness Ratio of schemes across different RTTs**

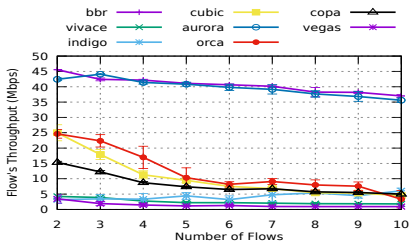



**Figure 14: Flow's throughput across diff. num. of competing flows**

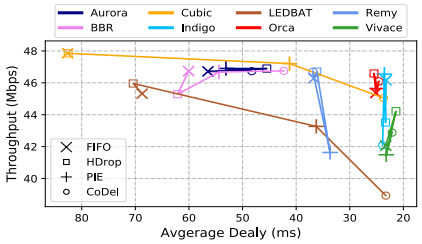



**Figure 15: Impact of various AQMs on diff. schemes**

schemes, we randomly choose one of the GENI servers and send traffic from one of our servers to it. Every 25 seconds, we add another flow to the network destined to the same GENI server and record their delivery rate and finish the test at 100s. We repeat the tests 5 times. Fig. 12 shows the average throughput of different flows for each scheme through time. As Fig. 12 illustrates, most of these schemes achieve appropriate fairness except Aurora. We observed that Aurora is very aggressive, to the degree that it won't let other competing Aurora flows get any share of the bandwidth. Orca's fairness property is based on two reasons: 1) choosing a well-studied objective of maximizing Power which is known to be optimal for the entire network [18, 30] as the basis for DRL-Agent's reward; 2) the fair AIMD nature of the underlying TCP scheme (Cubic) [12].

## 5.6 Deployment Friendliness

So far, we have shown that Orca performs very well in different environments. Does it get its high performance by penalizing (being aggressive toward) other competing TCP flows in the network? To answer this question, we set up two experiments. We use Cubic, which is default TCP in most of today's Operating Systems, as the base TCP. In the first experiment, in addition to a Cubic flow, we send another flow using a scheme under the test over 48*Mbps* outgoing link and $1 \times BDP$ buffer. We repeat the test for various RTTs and record the delivery rate of both flows. We run each test 5 times and report the average and standard deviation of friendliness ratio defined by $\frac{\text{delivery rate of CC scheme}}{\text{delivery rate of Cubic flow}}$ in Fig. 13. Large values of friendliness ratio show that the scheme is aggressive toward TCP, while the value of 1 means perfect TCP friendliness. As results show, in contrast with BBR and Aurora, Orca achieves good TCP friendliness which answers the raised question. Performance benefits of Orca do not come with the cost of penalizing other flows in the network, while the performance of schemes such as BBR comes with that cost. In another experiment, we increase the number of TCP (Cubic) flows and record the impact of the number of competing flows over the performance of each scheme. Here, we fix RTT to 40ms and to potentially handle more flows in the network, we increase buffer size to $2 \times BDP$. The throughput of the CC schemes under the test across the different number of competing Cubic flows are shown in Fig. 14. As the number of competing Cubic flows increases, the share of the scheme under the test should decrease. Results in Fig. 14 show that Orca decreases its share when new competing flows come to the network. However, schemes such as BBR and Aurora stay very aggressive toward other competing flows and achieve higher throughput than all other competing flows combined!

## 5.7 Impact of AQM on Orca's Performance

Generally speaking, TCP is a fully end-to-end distributed design that has no knowledge about the network and protocols that are used in the network including different AQM schemes used per switch/router on the path of TCP flows.

That lack of knowledge is part of the beauty of TCP and at the same time, it is part of the difficulty of designing a TCP scheme. A simple approach to managing this lack of knowledge (as classic CC schemes do) is to have certain assumptions about the network. However, these assumptions, if not met, can significantly impact performance. On the other hand, one of the advantages of learning-based approaches such as Orca is that they do not have/need any assumption about the network, how packets are delivered, or how the packets are dropped in the network. Instead, they consider the environment as a black-box that reveals itself only through observable end-to-end statistics and they learn how to react to them. To show this advantage of Orca, we set up an experiment. We connect a server to a client through a switch and an emulated 48*Mbps* link. We change among four different AQM designs including tail drop queue (FIFO), head drop queue (HDrop), PIE [45], and CoDel [44] in the switch. For any AQM scheme, we send one flow using different CC schemes for 60 seconds and record delivery rate and average one-way delay of packets for that flow. We repeat the tests 5 times. The average delivery rate and one-way delay of packets over all runs are shown in Fig. 15. As expected, Orca is insensitive to the AQM algorithm used in the switch and consistently achieves high performance. For classic approaches such as Cubic and LEDBAT, AQM sensitivity is high, while based on what we described learning-based designs generally show lower sensitivity. Insensitivity of Orca to AQMs provides a great opportunity to simplify design of new AQMs or use only simple AQM schemes in the network.

## 6 DISCUSSION AND FINAL NOTE

**How much general is Orca's model?** There is still much unknown about deep neural networks and reinforcement learning [31, 48]. Among them is the problem of *generalization*. Although Orca's performance results over complex unseen network scenarios presented in this paper (including inter-continental, intra-continental, and cellular environments, etc.) are promising and suggest that Orca learned a general policy, understanding, formulation, measurement, and improving generalization in the context of RL (and deep learning) and the question of whether a learned model can be successfully transferred are still active research topics in machine learning community [4, 6, 13, 15, 22, 41, 58].

**Does Orca provide performance guarantees?** The pragmatic two level-control in Orca can significantly alleviate performance issues in unseen scenarios. However, there is no guarantee that Orca will always perform well in every unseen scenario. After 3 decades of active research, none of the proposed CC designs can provide performance guarantees over uncertain network conditions of the Internet. That is partly due to the mentioned discussion about the nondecentralized nature of Power (see section 3.2). Also, Internet and its underlying packet-switch architecture are evolved on the basis of having a *best-effort* nature. That being said, the essence of setting the objective of achieving/having performance guarantees for a CC scheme which at the end of the day runs on a best-effort medium (Internet) is debatable.

**What if diversity is not a concern?** Even when the key concern is improving the performance in *a* certain network, our framework still can be used. Orca can learn proper model corresponding to a certain desired network when it is only trained over that environment. Besides, the underlying TCP in Orca can be selected among the ones that achieve higher performances in that environment.

**Limitations and future work:** Maximization of Power, which leads to optimal performance [18, 30], is a well-studied objective in the context of CC. However, one can argue that in *ultra-low latency* networks, the definition of Power practically favours the delay over the throughout. For instance, considering Power, in a network with 0.1ms *mRTT* and *X*bps bandwidth, a CC scheme gaining $\frac{X}{2}$bps throughput and 0.1ms delay is 5× more favourable compared to the one gaining *X*bps throughput and 1ms delay, while a throughput-oriented application might likely favour the 2nd scheme over the 1st one. This concern is less relevant on the Internet (with a few tens of milliseconds natural delay) for which Orca is designed, but it is worth mentioning that a more general form of power defined as $\frac{\text{Throughput}^{\kappa}}{\text{Delay}^{\xi}}$ can be employed to address this concern, when $\kappa$ and $\xi$ are selected properly.

Moreover, there are a few more interesting but likely more complicated approaches to potentially address the problem of possible different objective *preferences* of applications. Here, we briefly mention one of them and leave the details to future work. Can Orca learn to adapt to not only different environments but also different application objectives? In other words, can we learn a single policy network that is optimized over the entire (or appropriate portion of) space of objective preferences in a domain? Essentially, this and similar questions are the theme of a subset of machine learning topic named multi-objective reinforcement learning (MORL) [50, 57, 63]. Is it feasible to have an MORL-based solution in the context of CC? If yes, how? If no, why?

**Final Note:** We think that the practicality of a CC solution should not be sacrificed at the cost of the revolutionary aspect of it. Orca's current promising performance results demonstrate that instead of putting aside decades of CC designs, classic TCP techniques can be of great help when wisely combined with advanced learning methods. We believe that Orca by no means is the end of the story. Instead, we hope that it can generate more discussion on the need for more pragmatic learning-based CC solutions and provide a new design direction for improving TCP.

# REFERENCES


[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Soheil Abbasloo, Yang Xu, and H. Jonathan Chao. 2019. C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements. *IEEE Journal on Selected Areas in Communications* 37, 4 (2019), 918–932.

[3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *ACM SIGCOMM CCR*, Vol. 40. ACM, 63–74.

[4] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. 2018. Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers. *CoRR* abs/1811.04918 (2018). arXiv:1811.04918 http://arxiv.org/abs/1811.04918

[5] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 329–342.

[6] Andre Barreto et al. 2017. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 4055–4065. http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning.pdf

[7] Richard Bellman. 1957. *Dynamic Programming* (1 ed.). Princeton University Press, Princeton, NJ, USA.

[8] Mark Berman et al. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61 (2014), 5 – 23. http://www.sciencedirect.com/science/article/pii/S1389128613004507 Special issue on Future Internet Testbeds â Part I.

[9] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. 1994. *TCP Vegas: New techniques for congestion detection and avoidance*. Vol. 24. ACM.

[10] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. 2005. Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 15–28.

[11] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5 (2016), 50.

[12] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems* 17, 1 (1989), 1–14.

[13] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying Generalization in Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 1282–1289. http://proceedings.mlr.press/v97/cobbe19a.html

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} Vivace: Online-Learning Congestion Control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 343–356.

[15] Chelsea Finn, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine. 2016. Generalizing skills with semi-supervised reinforcement learning. *arXiv preprint arXiv:1612.00429* (2016).

[16] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[17] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

[18] R. Gail and L. Kleinrock. 1981. An Invariant Property of Computer Network Power. In *Proceedings of the International Conference on Communications*. Denver, Colorado, 63.1.1–63.1.5.

[19] Alfred Giessler, J Haenle, Andreas König, and E Pade. 1978. Free buffer allocation—An investigation by simulation. *Computer Networks (1976)* 2, 3 (1978), 191–208.

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

[21] Tom Henderson, Sally Floyd, Andrei Gurtov, and Yoshifumi Nishida. 2012. *The NewReno modification to TCP's fast recovery algorithm*. Technical Report.

[22] Irina Higgins et al. 2017. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1480–1490.

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[24] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933* (2018).

[25] Van Jacobson. 1988. Congestion avoidance and control. In *ACM SIGCOMM CCR*, Vol. 18. ACM, 314–329.

[26] Jeffrey Jaffe. 1981. Flow control power is nondecentralizable. *IEEE Transactions on Communications* 29, 9 (1981), 1301–1306.

[27] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *International Conference on Machine Learning*. 3050–3059.

[28] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM computer communication review* 32, 4 (2002), 89–102.

[29] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[30] Leonard Kleinrock. 1978. On flow control in computer networks. In *Proceedings of the International Conference on Communications*, Vol. 2. 27–2.

[31] Yan Lecun. 2017. My take on Ali Rahimi's "Test of Time" award talk at NIPS. (2017). https://www.facebook.com/yann.lecun/posts/10154938130592143

[32] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[33] Yuliang Li et al. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*. 44–58.

[34] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015). arXiv:1509.02971

[35] Shao Liu, Tamer Başar, and Ravi Srikant. 2008. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation* 65, 6-7 (2008), 417–440.

[36] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. 1995. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review* 102, 3 (1995), 419.

[37] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In

*Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[38] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 537–550.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[41] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2018. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347* (2018).

[42] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. 2015. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296* (2015).

[43] Ravi Netravali, Anirudh Sivaraman, Keith Winstein, Somak Das, Ameesh Goyal, and Hari Balakrishnan. 2014. Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement. (2014).

[44] Kathleen Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (2012), 42–50.

[45] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*. IEEE, 148–155.

[46] Shinik Park, Jinsung Lee, Junseon Kim, Jihoon Lee, Sangtae Ha, and Kyunghan Lee. 2018. ExLL: an extremely low-latency congestion control for mobile cellular networks. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. ACM, 307–319.

[47] Barak A Pearlmutter. 1995. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural networks* 6, 5 (1995), 1212–1228.

[48] Ali Rahimi and Ben Recht. 2017. Back when we were kids. (2017). https://www.youtube.com/watch?v=Qi1Yry33TQE "Test of Time" award talk at NIPS.

[49] Roger Ratcliff. 1990. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review* 97, 2 (1990), 285.

[50] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48 (2013), 67–113.

[51] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. 2010. LEDBAT: The New BitTorrent Congestion Control Protocol.. In *ICCCN*. 1–6.

[52] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[53] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[54] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms.

[55] R.S. Sutton and A.G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press. https://books.google.com/books?id=uWV0DwAAQBAJ

[56] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *Proceedings-IEEE INFOCOM*.

[57] Kristof Van Moffaert and Ann Nowé. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research* 15, 1 (2014), 3483–3512.

[58] Huan Wang, Stephan Zheng, Caiming Xiong, and Richard Socher. 2019. On the Generalization Gap in Reparameterizable Reinforcement Learning. *CoRR* abs/1905.12654 (2019). arXiv:1905.12654 http://arxiv.org/abs/1905.12654

[59] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. ACM. https://doi.org/10.1145/2486001.2486020

[60] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks.. In *NSDI*. 459–471.

[61] Lisong Xu, Khaled Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *Proceedings-IEEE INFOCOM*, Vol. 4. IEEE, 2514–2524.

[62] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 731–743.

[63] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. In *Advances in Neural Information Processing Systems*. 14610–14621.

## APPENDIX A RELATED WORK

More than three decays of active research in congestion control has brought us plethora of congestion control schemes including numerous variants of TCP. Here, we briefly overview some of them.

Early results during 1980s (e.g. [26]) indicated that achieving optimal point of operation in which all users get maximum throughput, minimum delay, and a fair share of the network resources in a fully distributed way is not feasible. These results motivated heuristic approach toward the design of TCP which primarily was intended to be a fully end-to-end and distributed algorithm for controlling congestion. Among the early schemes, TCP Tahoe, TCP Reno [25], and TCP NewReno [21] can be named. These schemes introduced the heuristic AIMD-based algorithms based on the consideration of loss of packets as the key indicator of congestion. Later, different schemes tried to replace the simple linear incremental functions of these old proposals. Among them, BIC [61] and TCP Cubic [20] can be mentioned. For example, Cubic which is today's default TCP in most of the platforms, replaced the incremental function of classic AIMD-based approaches with a cubic function.

While these schemes hard-wire the loss of packets to certain control actions, another set of designs tried to find new ways of interpreting congestion in the network. The key concern was that the schemes that wait for loss of a packet to react to congestion, basically make congestion to detect congestion. TCP Vegas [9] was among the early schemes that introduced the use of delay as a congestion signal. Using delay, there was a chance to detect future congestion without making congestion. Designs such as Compound TCP [56] and TCP Illinois [35] are among the schemes that followed this delay-based approach and combined it with the loss of packets as congestion indicators. Among some recent delay-based designs LEDBAT[51] and Copa [5] can be named.

Some other proposals attempted to laser-focus on a certain environment and use some prior information about that environment during the design of CC. For instance, schemes such as Sprout [60], ExLL [46], and C2TCP [2] target only cellular networks and employ some of the unique characteristics of these networks during the design of CC, while DCTCP [3], Timely [38], and HPCC [33] target datacenter networks. Another example is BBR [11] which first targeted inter-datacenter communications, though later Google tried to push the usage of it in other environments. BBR uses a white-box approach and models the network in a certain way. Using that model, it avoids to interpret loss of packet as a congestion signal. Instead, it keeps sending $2 \times BDP$ of packets to the network, while continuously attempts to measure the $BDP$ of network. This lack of dependency to packet loss helps it to achieve high throughput. However, this as we have shown, makes serious TCP friendliness issues for it. The key problem with these schemes is that they either hard-wire some events to some control actions or assume that network will always follow certain simple models.

Remy [59] pushes the hard-wire mapping of events to actions done in classic TCP to a new extreme and attempts to prepare an offline calculated mapping from all possible events to good actions by walking through all possible events in a brute-force manner. In other words, it attempts to learn best event-action mappings by walking through all of them. This offline-optimization approach leads to a major key practical issue. Remy requires accurate assumptions about the network in advance to learn the mapping. However, it is not feasible to make always correct assumptions about the network environments including number of competing flows, max link speeds, delay of network, etc, especially when the network is the Internet.

To address the issues of hard-wiring controlling actions to events and use of the certain models for the network, some work attempt to use more advanced learning-based designs. PCC-Vivace [14] proposes to use online learning techniques to choose the best sending rates. Indigo [62] uses imitation-learning while Aurora [27] leverages vanilla DRL to determine the sending rates and replace the TCP. Although these schemes potentially can do a better job for controlling congestion without tying predefined control actions to predefined input events, as we mentioned in this paper, they experience performance issues and problems in practice.

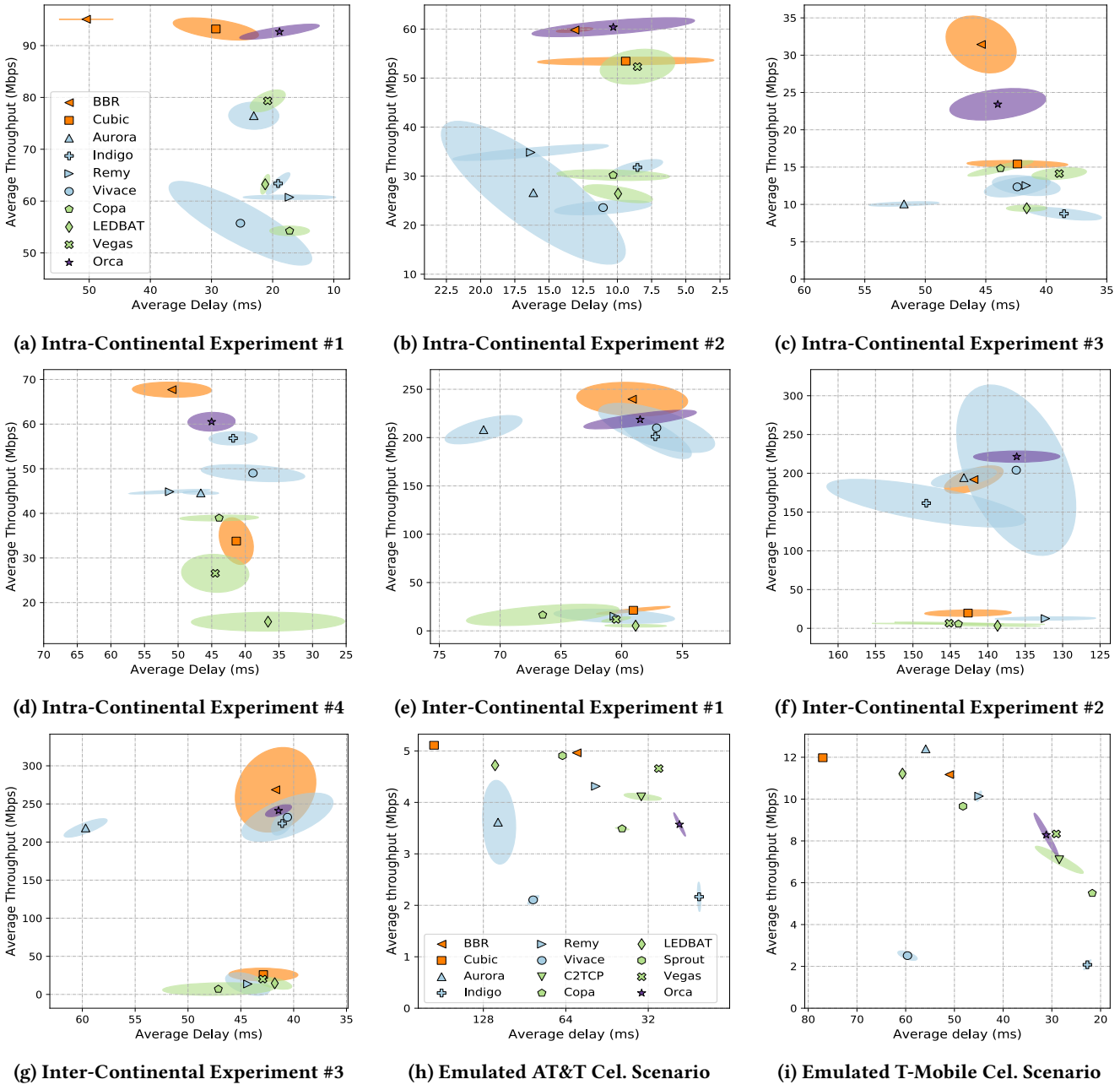
There is also another category of CC solutions that do not see TCP as a fully end-to-end approach. Schemes in this category argue that since fully end-to-end TCP faces distributed measurement issue and does not have enough information about the network to take a proper action, it is better to use explicit feedback from the network to set the sending rates. XCP [28] and RCP [10] are among the schemes using this philosophy. However, the key issue with these designs is that they are not deployment-friendly, because these schemes require new switch/router hardware, protocols and collaboration of all entities on a path which is not a feasible approach especially for the Internet.

## APPENDIX B SAMPLE RESULTS

Here, we choose some samples of the client-server connections from our global testbed corresponding to the experiments done in section 5.1 and show the results gained for each of them in Fig. 16. In addition, we demonstrate the detailed version of highly variable link emulations done using AT&T and T-Mobile cellular traces.

## APPENDIX C TRAINING'S SETTING

Orca's policy networks, taking state $s$ and outputting action $a$, use two hidden layers with 256 units for both layers. The output layer is a tanh layer to bound the actions. The critic networks, taking state $s$ and action $a$ and outputting $Q$ values, have two hidden layers with 256 units for both layers. The action is not included until the second hidden layer, and the output layer is a linear unit representing the $Q$ function. All hidden layers in policy and critic networks are followed by Leaky ReLU nonlinearity. Table 3 shows other parameters used during the training of Orca.




(a) Intra-Continental Experiment #1 (b) Intra-Continental Experiment #2 (c) Intra-Continental Experiment #3

(d) Intra-Continental Experiment #4 (e) Inter-Continental Experiment #1 (f) Inter-Continental Experiment #2

(g) Inter-Continental Experiment #3 (h) Emulated AT&T Cel. Scenario (i) Emulated T-Mobile Cel. Scenario

**Figure 16: Sample of the results for the experiments done in section 5.1 showing average throughput and average one-way delay gained by different schemes. Center of each ellipse shows the average value of each scheme, while the ellipse indicates the standard deviations from the average values of the delay and the throughput and their covariance (i.e., the shaded ellipses depict $1-\sigma$ variations across all runs). Note that the scale of x/y axes are different in different graphs.**

**Table 3: Parameters Used During the Training**

| The Parameter | Value |
|---|---|
| Optimizer | Adam |
| Actor's Learning Rate $\eta$ | 0.0001 |
| Critic's Learning Rate $\rho$ | 0.001 |
| Batch Size $N$ | 8096 |
| Target Networks Updating Coefficient $\tau$ | 0.001 |
| Actor's Exploration Noise | $\mathcal{N}(0, 0.2)$ |
| Recurrent Dimension | 10 |
| Discount Factor $\gamma$ | 0.995 |
| Total samples from Each Actor @ Each Round | 50K |
| $\zeta$ | 5 |
| $\beta$ | 1.25 |