Learning-Based vs Human-Derived Congestion Control: An In-Depth Experimental Study

Mihai Mazilu, Luca Giacomoni and George Parisis
School of Engineering and Informatics, University of Sussex
{m.mazilu, l.giacomoni, g.parisis}@sussex.ac.uk

***Abstract*—Learning-based congestion control (CC), including Reinforcement-Learning, promises efficient CC in a fast-changing networking landscape, where evolving communication technologies, applications and traffic workloads pose severe challenges to human-derived, static CC algorithms. Learning-based CC is in its early days and substantial research is required to understand existing limitations, identify research challenges and, eventually, yield deployable solutions for real-world networks. In this paper, we extend our prior work and present a reproducible and systematic study of learning-based CC with the aim to highlight strengths and uncover fundamental limitations of the state-of-the-art. We directly contrast said approaches with widely deployed, human-derived CC algorithms, namely TCP Cubic and BBR (version 3). We identify challenges in evaluating learning-based CC, establish a methodology for studying said approaches and perform large-scale experimentation with learning-based CC approaches that are publicly available. We show that embedding fairness directly into reward functions is effective; however, the fairness properties do not generalise into unseen conditions. We then show that RL learning-based approaches existing approaches can acquire all available bandwidth while largely maintaining low latency. Finally, we highlight that existing the latest learning-based CC approaches under-perform when the available bandwidth and end-to-end latency dynamically change while remaining resistant to non-congestive loss. As with our initial study, our experimentation codebase and datasets are publicly available with the aim to galvanise the research community towards transparency and reproducibility, which have been recognised as crucial for researching and evaluating machine-generated policies.**

I. Introduction

Reinforcement learning (RL) has gained substantial momentum in recent years, and increasingly sophisticated systems that span a wide array of applications, such as games [1]–[4], natural language processing [5], and even nuclear fusion [6], have been developed. Therefore, it has only been natural to see RL making its way into the networking realm, specifically in routing [7], video rate control [8], [9], network access [10], [11], security [12], [13] and proactive caching [14], [15]. Congestion control (CC) has recently been rethought through the lens of RL, promising efficient CC where fast-evolving communication technologies, applications, and traffic workloads pose substantial challenges to human-derived CC algorithms, such as Cubic [16] and BBR [17]. Other learning-based CC approaches, such as Remy [18], PCC Allegro [19] and PCC Vivace [20], have also been explored.

It has been widely accepted that experimentally evaluating CC algorithms is a challenging task due to the complexity in designing experiments that exercise CC in a wide range of network conditions and the breadth of CC performance metrics that must be taken into account in unison, and not in isolation [21]. Learning-based CC approaches pose additional challenges because of the black-box nature of decision-making (e.g., when deep RL is employed) or when the state space gets very large to track decision making (e.g., in Remy). This renders machine-derived CC policies difficult to reason about. The issues identified above are exacerbated by the diversity in the implementation of learning-based approaches and CC algorithms. For example, Remy [18], TCP-Drinc [22] and SmartCC [23] are implemented as simulation models. On the other hand, Aurora [24] are trained within a simulation environment and subsequently integrated into a user-space prototype. Orca [25], Astraea [26] and Spine [27] perform RL model training and inference within a user space application, but the core of the CC is integrated within the Linux kernel. Sage [28] is trained over pre-collected traces from a diverse set of CC algorithms before deployed within the Linux kernel. PCC-Vivace comes with both a user- and kernel-space implementation, which behave very differently to each other.

We posit that existing literature on learning-based CC falls short when it comes to studying the behaviour of the CC policies and evaluating their performance; reproducibility of results has been underthought. Evaluation methodologies have been ad-hoc with experimentation conducted on emulated and 'in the wild' environments. Network emulation can be very effective when experimenting with network protocols, but it is crucial that complexities and limitations of the underlying queuing disciplines, buffering, hardware offloading and associated CPU overhead [29] are discussed. In Aurora [24], Orca [25] and QTCP [30] experimentation involves sending a single flow into a single path, and CC is only exercised when/if the sender's sending rate is higher than that capacity of the emulated path. In such scenarios, the congestion window (or sending rate) is quickly set to a value that the underlying RL agent considers (potentially locally) optimal and does not change much after that. Such experiments show how effective a CC algorithm is in capturing the available bandwidth in the absence of contending flows but fall short when it comes to fairness and showing responsiveness in the face of hotspots. In [25], experimentation with different bandwidth-delay product (BDP) paths is un-systematic, with multiple BDP values tested when evaluating strengths in terms of eliminating bufferbloat, but only a single value when evaluating friendliness with TCP. 'In the wild' environments (e.g., using GENI, and/or EC2 servers), are suitable for showcasing the feasibility of a CC approach and its compatibility with legacy CC, but they pose profound limitations when it comes to interpreting and

reproducing results [31]. For example, in [25] a single flow is sent through inter- and intra-continental paths but it is unclear if the flow experiences any congestion at all. When evaluating fairness, the characteristics of the randomly selected paths are not discussed so it is impossible to reproduce the results. In [26], Astraea convergence ability appears to be exceptional when evaluating within the parameters used to train the underlying model, however, in the code provided by the authors, we have observed that fair queuing has been configured in the emulated links; this skews the observed behaviour, which, as shown in this paper, is very good but not exceptional. Finally, it is worth noting that that there has not been any substantial comparative studies using simulations models and learning-based CC approaches. Frameworks like [32] and [33] open opportunities for developing an experimentation framework for RL-based CC using simulations.

In this paper, we expand on and update our initial study published in [34] and present a reproducible and systematic study of learning-based CC with the aim of highlighting advances and uncovering fundamental limitations of the state-of-the-art. We identify challenges in evaluating learning-based CC, and devise a methodology and set of benchmark experiments to examine efficiency, fairness, and responsiveness in single- and multi-bottleneck topologies. We perform large-scale experimentation with publicly available RL-based CC models, namely *Orca* [25], *Sage* [28] and *Astraea* [26], and *PCC Vivace* [20], a learning rate control algorithm. We compare those with *Cubic* and *BBR version 3*, both being human-derived and interpretable CC policies.

Three themes emerge from our expanded evaluation; (1) we have successfully reproduced Orca's behaviour, matching earlier findings across different hardware configurations, with only limited variation; (2) we showcase that current state-of-the-art RL approaches continue to struggle at *capturing available bandwidth* in the presence of base RTT fluctuations compared to human derived approaches; (3) generalisation breaks down: Astraea's fairness holds only within its training parameters and degrades sharply beyond them, while Sage becomes unstable outside its training range, oscillating its sending rate leading to slow convergence.

II. Challenges

Previous work has identified several pitfalls in evaluating CC algorithms [21] such as conducting experiments that do not actually exercise congestion control, focussing on specific performance measures, and evaluating CC in a narrow range of network parameters. In this paper, we expand on challenges that are specific to learning-based CC, namely the (1) machine-generated nature of the CC policy which often results in uninterpretable CC behaviour, (2) wild heterogeneity in the implementation of RL-based CC, and, (3) evaluating proposals within a single testbed, while controlling trade-offs related to reproducibility, fidelity, and representativeness of results.

Uninterpretable CC behaviour. In contrast to human-generated protocols, RL-based CC protocols are inherently black-boxed. Traditional CC behaviour is interpretable and substantial research has been dedicated to studying CC analytically [35], [36] and experimentally [21], [37]. RL-based CC is driven by an underlying policy (deterministic or stochastic) that is learned by interacting with a real, emulated or simulated network, or pre-existing captured traces. The decision-making process that drives the evolution of the transmission rate (and/or congestion window) is embedded within intricate and unintepretable parametric models. Experimentally studying such RL-based CC must be done extremely carefully; one can identify patterns and provide plausible explanations but, at the same time, must be wary of the black-box nature of the underlying CC policy. For example, in Orca the authors argue that fairness arises due to the inclusion of the power metric and additive-increase multiplicative-decrease (AIMD) into the decision making [25]. However, in Section IV-A, we show a plethora of experimental setups where Orca is extremely unfair, which puts the original fairness arguments in question. Similarly, Astraea appears to be performing exceptionally well in terms of convergence in the experimentation shown in [26]. However, in Section IV-E we show that Astraea's convergence profile is much affected by fair queue configuration that has been part of the experimentation in [26].

Heterogeneity in Implementations. As there is no agreed mechanism (nor any standardised operating system support, e.g., by exposing a CC API to a learning framework) to implement RL-based CC, existing proposals are based on ad-hoc prototypes implemented as simulation models, user-space applications, kernel code, or combinations of these. For example, Aurora [24] is trained within a bespoke simulator and the learned model is integrated into a user-space prototype built with UDT [38]. Orca, Sage Astraea, on the other hand, are built within the kernel (side to side with Cubic), combined with a user-space component that is performing learning and inference, with the two communicating through shared memory. Sage learns through pre-collected traces of human-derived CC algorithms, and performs inference similarly to Orca. Comparing RL-based CC approaches in a meaningful and reproducible way is far from trivial. As an example, Aurora is implemented in the user space and is known to be very CPU-intensive; however, this should not be attributed to the RL components of the approach (at least not any more than other similar approaches), e.g., as it has been in [27].

Reproducibility, Fidelity and Flexibility. Experimental evaluation of RL-based CC approaches involves several trade-offs. Simulation-based evaluation enables reproducible results and flexibility in designing a wide variety of experiments. However, fidelity of results is degraded substantially, even when state-of-the-art packet-level, discrete event simulators, such as ns-3 or OMNeT++, are used. 'In the wild' experimentation can only be done with prototypes that are built within the network stack of an operating system, but enables high fidelity. On the other hand, reproducibility is problematic because several parameters (e.g., network topology, routing, background traffic) may be unknown or out of control, when conducting an experiment. Flexibility is also limited as it is extremely difficult to access network environments that are representative of the variety present in the real world. Network emulation offers a middle ground for these trade-offs. Fidelity is substantially higher compared to a simulated environment but there are limitations related to CPU overhead, hardware

offloading and attached queuing disciplines. Reproducibility of results is possible if all network parameters are recorded - nondeterminism associated with running a prototype on a real system can result in different outcomes, so it is important that multiple runs per experiment are included to asses the level of variability for each measured metric. Network emulation provides substantial flexibility in experimentation, but this needs to be balanced with fidelity-related constraints.

III. Methodology

In this study, we adopt an emulation-based approach for two reasons; (1) seminal work on learning-based CC has yielded open-source prototypes (see Table I in Section V) that can be tested on real-world and emulated networks; (2) we aim to maximise fidelity, reproducibility, and flexibility, therefore 'in the wild' experimentation was deemed unsuitable.

Selected CC Approaches. In Section V, we briefly discuss all different RL-based CC approaches, categorising them into clean slate and hybrid ones. In this study, we expand on both the learning-based and baseline algorithms, compared with our earlier work in [34]. More specifically, we include Orca [25], Sage [28] and Astraea [26], as RL-based CC algorithms, and PCC Vivace [20], as an interpretable learning rate control algorithm. We compare these approaches with Cubic and BBRv3, as they involve both loss-based (Cubic and BBRv3) and delay-based (BBRv3) traits.[1] We have dropped Aurora from our experimentation for clarity in all presented plots, because we have studied it extensively in [34] where it became apparent that its performance is very poor (also shown in [26]) in real-world network setups, particularly when multiple flows compete with each other or TCP flows. Aurora senders aggressively fill up all available network bandwidth dwarfing other Aurora and non-Aurora flows. Unfortunately, although we would have liked to include work, where learning is done in conjunction with an expert (e.g., as in [41], [42]), source code for these is not publicly available. For Orca, Sage and Astraea, we utilise the pre-trained models provided by the authors. Finally, for PCC Vivace, we use its user-space implementation because the kernel-based one generated inconsistent results that could not be attributed to Vivace's key characteristics.

Performance Metrics. As the implementations of the selected CC approaches are different from each other, not all performance metrics are collected in the same fashion for all approaches. *Application goodput* was measured differently between them. For Orca, Astraea and Sage, goodput measurement was embedded in the receiver included in the authors' source code. Vivace's goodput was measured using the UDT [38] data collection API. For Cubic and BBRv3, we used *iPerf3* [43] to measure goodput. For the *round trip time* (RTT) and *congestion window*, we used *socket statistics* [44] to log the values directly from the socket state for BBRv3, Cubic, Orca and Sage. For Astraea, we recorded the RTT and congestion window through its client application. For Vivace, we used the UDT API to measure the RTT and sending rate; Vivace does not have a congestion window. To record *retransmissions* and *link utilisation*, we used *sysstat* [45].

Experimental Setup. Our emulation environment is built using *Mininet* [46]. All network nodes (hosts and routers) are represented as network namespaces in our Linux-based emulation host with the following specs: AMD Ryzen Threadripper PRO 7965WX 48 core processor, 64 GB of memory, running Ubuntu 22.04 LTS using the BBRv3 enabled 6.4.0 kernel provided by [47], patched with the kernel components required for Astraea and Orca to work. Sage authors only provide a pre-patched and pre-compiled kernel (version 4.19) which we were unable to run on bare metal due to hardware compatibility issues. We therefore have run Sage separately on top of a Hyper-V hypervisor to incur the least overhead.[2]

Experimentation is performed in emulated dumbbell and parking lot topologies. We used *netem* to emulate propagation delay and a token bucket filter to regulate transmission rate at bottlenecks. All other network interfaces are deployed without any propagation delay or rate limiter in place. Hosts are configured to be senders or receivers in only a single data flow at any given time, and we set their send and receive buffers to a large value so that the bottleneck is always in the network (and not on the sending host or as a result of flow control) for all the different buffer capacities tested in this paper. Queuing at the bottleneck interface is tail drop. To avoid emulation artifacts related to hardware offloading (e.g. TCP segment offloading is performed when forwarding segments, adding unnecessary latency), we have disabled hardware offloading altogether and experimentally verified that the resulting induced CPU overhead does not affect our experiments. Each experiment is run five times, and we report average values and standard deviation in error bars or shaded areas.

IV. Experimental Evaluation

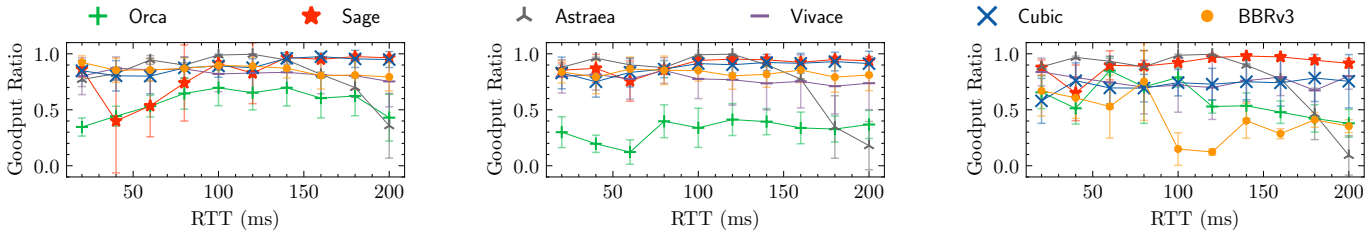
In this section, we present the results of our 404-hour long experimentation, during which 6270 Orca, Sage, Astraea, Vivace, Cubic, and BBRv3 flows were measured. We have collected metrics related to network interfaces (e.g., utilisation, retransmissions), CPU and memory parameters (e.g., CPU load and memory usage), and the transport layer (e.g., congestion window, round trip time). We have made the codebase and configuration files required for reproducing the results, and the extracted dataset available on *figshare* [48] and code at [49].

A. Fairness

Fairness in sharing a bottleneck's bandwidth is a very important property that CC algorithms must have. AIMD CC has been shown to yield a fair allocation of bandwidth to competing flows with the same RTT, but it is problematic when competing flows experience different RTTs. To date, little is known about the fairness of learning-based CC; here,

[1] BBRv3 has not been studied much beyond [39], [40], therefore, this study also contributes in better understanding its behaviour in a wide range of network parameters and single- and multi-bottleneck topologies. Since our main focus is on learning-based CC, we omit profiling the myriad of TCP CC variants available in the literature.

[2] We are confident that the virtualisation overhead does not affect Sage's performance because we have also run all other approaches on top of the hypervisor and the results were consistent with the ones produced when run on bare metal and presented here.



(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 1: **Intra-RTT Fairness**. Goodput ratio for two competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps, both flows experience the same base RTT (shown on x-axis), buffer capacity is set to 0.2× (a), 1× (b), and 4× (c) the BDP.

we focus on the effect that RTT, available bandwidth, and buffer capacity have on fairness when two flows contend for bandwidth. The first flow runs for 2000× its *base RTT* (i.e., the delay a packet experiences when the network is empty); the second flow starts after the first one has been running for 25% of its full duration.

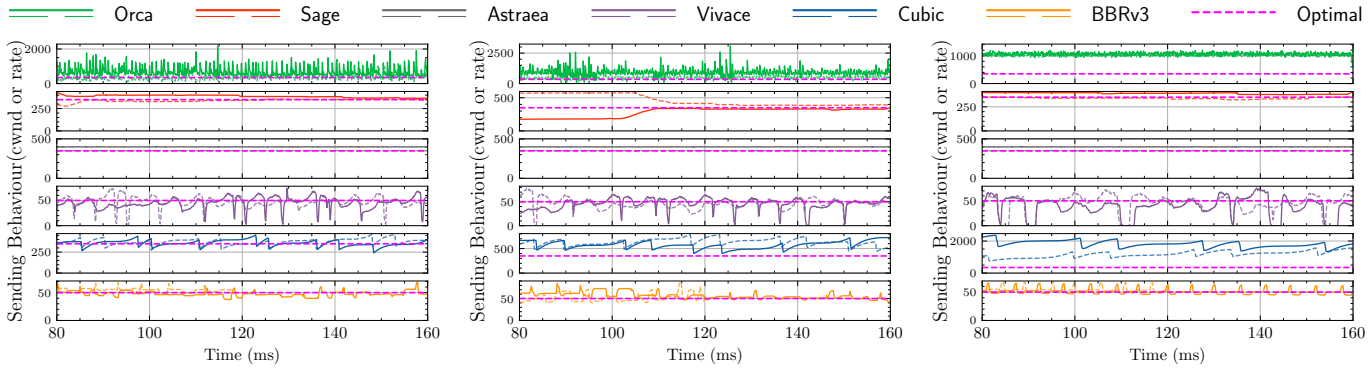
1) Fairness and RTT Variation: First, we examine how the base RTT that flows experience affects the fairness profiles of the selected CC approaches. We differentiate between the *intra-RTT* and *inter-RTT* cases, where flows experience the same or different base RTT values, respectively. The bottleneck bandwidth is set to 100Mbps.

Intra-RTT Fairness. Figures 1a-c show the goodput ratio of the flows for when the buffer capacity is set to 0.2×, 1× and 4× their path's BDP, respectively. Flows experience the same base RTT, reported on the x-axis. We report results for the last 500 RTTs of the experiment so that all CC schemes have the chance to converge to a stable bandwidth allocation.

Orca promises fairness by (1) integrating power in its reward function and (2) AIMD in its operation. Orca yields a substantially more unfair allocation of bandwidth compared to Cubic when the buffer capacity is set to 1× the BDP (Figure 1b), with goodput ratios consistently under 0.5; i.e., one flow grabbing at least more than double the bandwidth allocated to the other one. When the queue size is 0.2× the BDP, Orca performs much better, yet it is still more unfair than Cubic. With the buffer capacity set to 4× the BDP (Figure 1c), Orca's fairness improves, but it outperforms Cubic only for some RTT values, remaining less fair otherwise. Sage maintains high fairness for most tested queue sizes. Sage is trained using a pool of policies, split across two sets containing traces of existing schemes found in the Linux kernel (including BBRv2, Cubic and Vegas). Set 1 encompasses single flow scenarios with changing network conditions. Set 2 is aimed at providing Sage with observations of TCP friendliness. The schemes used to train Sage are effective when it comes to intra-RTT fairness and, expectedly, Sage performs very well in this setup. It is important to note that, while Sage's training bandwidth and base RTT parameter spaces are within the range we use here, the selected queue size of 0.2× the path's BDP is not; this is the likely cause of Sage being less fair (and with high variance) in several of the tested RTT values (see Figure 1a). Astraea, the first RL-based CC scheme to introduce fairness directly in its reward function, in combination with multi-agent RL, maintains high fairness across the parameter space in which it is trained (10 to 140 ms of RTT). When both of the flows in the experiment are outside the training parameters, we see that fairness tapers off, as seen from the data points in the range of 160 - 200 ms in Figure 1, a trend that we observe throughout this study. The authors of Vivace provide a theorem, stating that when any number of senders share a bottleneck link and all senders share the Vivace utility function, the sending rates converge to a fair configuration. As we see in Figures 1a - 1c, Vivace achieves high fairness (but not higher than TCP Cubic) across the base RTTs for all tested queue sizes. Cubic maintains good fairness properties across the parameter space. When the queue size is 4× the BDP, fairness declines, and this is because Cubic flows converge very slowly to a fair share due to their large BDP values. BBRv3 fairness profiles when the queue size is 0.2× and 1× the path's BDP are very good. When the queue size is 4× the BDP, BBRv3 shows substantial unfairness. Having looked at individual BBRv3 runs, we observed that the second flow exits the start-up phase prematurely. Therefore, it cannot capture a fair share of the bandwidth, because the maximum observed bandwidth, which is used to calculate the path's BDP, is very low. This is because, when the second flow starts immediately after the probe RTT phase of the first flow, it cannot initially capture much bandwidth. Variability in the start time of the second flow, introduced by emulation noise, leads to these timing discrepancies. This trend was also observed in [39].

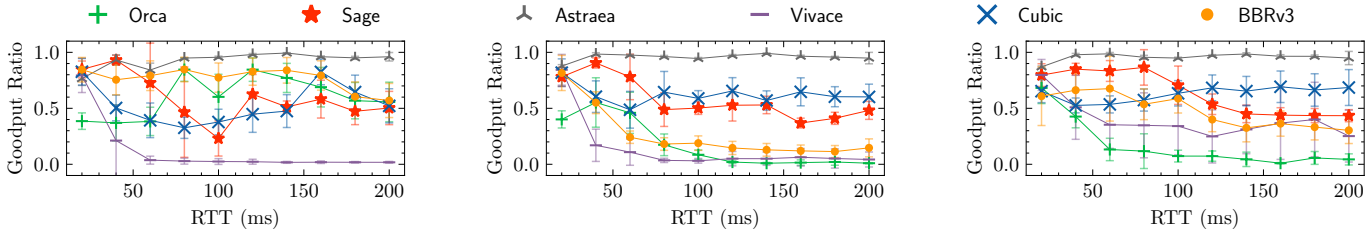
Congestion Window Evolution. To better understand fairness, we look at the evolution of the congestion window for competing flows (pacing rate for Vivace and BBRv3). Figure 2 shows this when the base RTT is 80ms; a value where Orca performs well when the buffer capacity is 4× the BDP. We plot results for the duration at which flows co-exist. We also plot the line denoting fair allocation that maximises network utilisation and minimises experienced latency; i.e., $\frac{BDP}{2}$ for Orca, Sage Astraea, and Cubic, and $\frac{bandwidth}{2}$ for Vivace and BBRv3. For BBRv3, we show the pacing rate, which is calculated using the *bytes in flight*, rather than the congestion window. This is because BBR purposely uses the congestion window as a limit to the value of bytes in flight, to combat ACK aggregation or compression [17], and therefore the congestion window is a much larger quantity than bytes in flight.

Orca's congestion window progression is noisy particularly in Figures 2a and 2b. This is because Orca issues coarse-grained congestion window updates at each monitoring inter-



(a) RTT: 80ms, Buffer Size: 0.2× BDP (b) RTT: 80ms, Buffer Size: 1× BDP (c) RTT: 80ms, Buffer Size: 4× BDP

Fig. 2: **Intra-RTT Fairness**. Congestion window (sending rate for Vivace and BBRv3) for two competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps, base RTT is 80ms, buffer capacity is set to 0.2×, 1× and 4× the BDP.



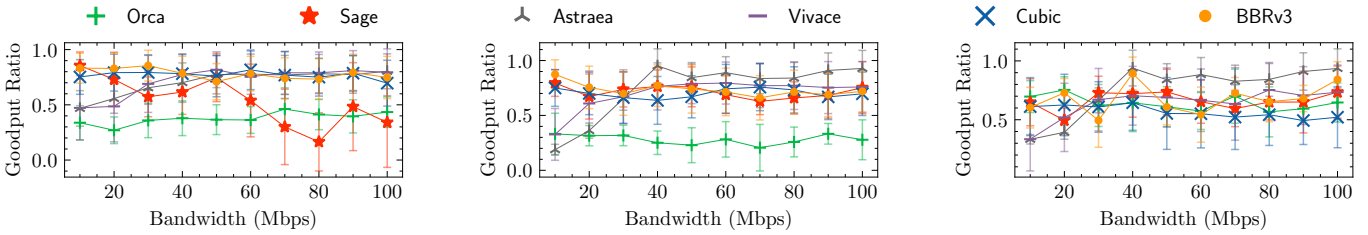
(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 3: **Inter-RTT Fairness**. Goodput ratio for two competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps and buffer capacity is set to 0.2× (a), 1× (b), and 4× (c) the BDP of the path with the smallest RTT. Flows experience different RTTs; RTT of first flow is set to 20ms and RTT of second flow is shown on x-axis.

val, while Cubic continuously performing per-ACK window adjustments [25]. When the buffer capacity is 0.2× the BDP (Figure 2a), the progression appears to be extremely noisy, with large spikes. When the buffer capacity is set to 1× the BDP (Figure 2b), values continue to be noisy, oscillating around an unfair allocation. Finally, when the buffer capacity is set to 4× the BDP (Figure 2c), Orca sets the congestion window to a fair allocation, closer to the optimal allocation compared to Cubic; maintaining high link utilisation and lower latency than Cubic. Sage's congestion window does not fluctuate much after convergence, which is otherwise slow. For example, when the buffer capacity is 1× the BDP (see red lines in Figure 2b), we see Sage converging to a fair and optimal allocation 110 seconds into the experiment. Looking at Astraea in Figure 2, we see how stable and close to optimal the congestion window is, where there is roughly only a 50 packet difference between the congestion windows of the two flows. We have observed this to be the case across all different queue capacities. The sending rate of Vivace fluctuates as it performs its probing trials, which are part of its gradient-ascent learning scheme. Vivace is quite noisy across the different buffer capacity values in Figure 2. When the queue size is set to 4× the BDP (Figure 2c), the reversal in the direction of the sending rate [20] leads to periodic under-utilisation. Cubic's congestion window evolution follows the known Cubic pattern, which yields a close-to-optimal allocation when the buffer capacity is 0.2× the BDP (Figure 2a). As Cubic is loss-based, it fills the buffer before loss occurs, and this is why the congestion window is much larger than the optimal one for larger buffer capacities (Figures 2b and 2c). The pacing rate of BBRv3 follows the optimal threshold well. In Figure 2b we observe BBRv3 initially overestimating the bottleneck bandwidth; then converging to a fairer allocation. Note that the probe RTT phase, with the characteristic drop in the number of segments sent in the network, is not visible in Figure 2. This is because BBRv3 limits its sending rate in the probe RTT phase by decreasing the congestion window, and not the pacing rate (which is what we plot here).

Inter-RTT Fairness. To assess fairness when competing flows have different base RTTs, we repeat the experiment described above but fix the base RTT of the first flow to 20ms and vary that of the second from 20ms to 200ms. In Figure 3, we report results for the last 500 RTTs of the experiment to let CC converge to a fair allocation.

Orca is generally fairer than Cubic, when the buffer capacity is set to 0.2× the BDP of the flows' path, more so as the difference between the base RTTs of the competing flows increases (see Figure 3a, from 60ms upwards). For larger buffer capacities (1× and 4× the BDP), Orca shows persistent substantial unfairness, which becomes more and more pronounced as the difference between the RTTs of the two flows increases (Figures 3b and 3c). Sage does not manage to consistently outperform Cubic, as none of the traces used in training include inter-RTT scenarios. Furthermore, the CC



(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 4: **Fairness with Bandwidth Variation**. Goodput ratio for two competing flows in a dumbbell topology. Base RTT is 40ms, bottleneck bandwidth varies as shown on the x-axis, buffer capacity is set to 0.2× (a), 1× (b), and 4× (c) the BDP.

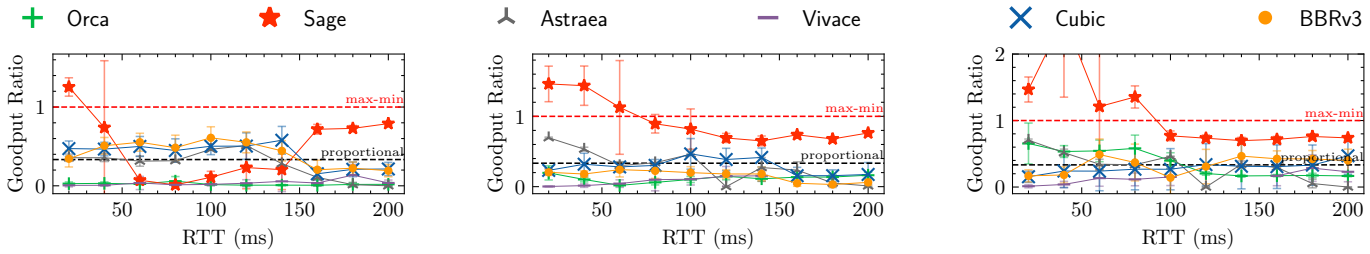
schemes used to train Sage lack heuristics that enable learning fairness in inter-RTT scenarios. In the lower range of base RTT values in Figure 3, we see that Sage is fairer than Cubic, but as the RTT increases beyond 100ms, Cubic performs better. Astraea performs the best among all tested CC approaches, achieving close to optimal goodput ratios across all tested RTTs and queue sizes. Astraea maintains high fairness even when one flow experiences RTT outside its training range 160 - 200 ms in Figure 3). Astraea employs an RTT-agnostic reward component to enforce fairness, which, in combination with a fixed monitoring time period (independent of any RTT values), appears to be very effective, as shown in our results. In [20], it is shown that when multiple Vivace senders compete over a single bottleneck link, their rates converge to the same fair value. However, it is not clear if the underlying assumption is that all flows experience the same RTT.[3] From our results, it becomes evident that Vivace is unfair when competing flows experience different base RTTs, for all tested queue sizes (see Figures 3a - 3c). Looking at individual runs, we discovered that the flow with the lower base RTT is the one dominating the bottleneck, and this is likely due to the fact that it is going through the gradient ascent cycle more often. In Vivace, the frequency of this cycle is based on the monitoring interval which is set to 1 RTT. Cubic is known to be unfair when competing flows experience different base RTTs [16] and this is evident in Figure 3, particularly when the buffer capacity is 0.2× the BDP (Figure 3a). BBRv3's inter-RTT fairness varies greatly with buffer size. When the queue capacity is set to 1× BDP (Figure 3b), the flow with the larger base RTT sustains a higher in-flight byte volume and starves the other flow. Increasing the queue capacity (see Figure 3c) results in BBRv3 being fairer. This is because the pacing gain increases from 0.75 to 0.91 during the bandwidth probe down phase [50] and this makes the existing flow yield less of its bandwidth to the joining flow. When the buffer is very small (Figure 3a), the higher RTT flow cannot claim a substantially higher buffer occupancy than the 20ms flow, resulting in a fairer allocation than when the buffer capacity is 1× the path's BDP.

2) Fairness and bandwidth variation: we experiment using the same setup as above, but we vary the provisioned bandwidth and fix the base RTT value to 40ms. Figure 4 shows the measured goodput ratio for the three different buffer capacity values studied above. When the buffer capacity is set to 0.2× (Figure 4a) and 1× (Figure 4b) the BDP, Orca is consistently less fair than Cubic. When the buffer capacity is large (Figure 4c), Orca is fairer to Cubic for several values of the tested bandwidth. For Sage, this experimental setup is not one where the Linux implemented schemes included in its training pool struggle, so it performs well, unable though to meaningfully outperform Cubic. Astraea performs the best across all schemes within its training parameters (40 to 100 Mbps); however, it cannot generalise to bandwidths outside its training range, resulting in a less fair profile. Its fairness remains the highest even when the queue size is 4× the BDP as in Figure 4c. Vivace produces an unfair profile for low bandwidth values, as evidenced in Figure 4 (particularly 4b and 4c). We believe this is because for such low values, Vivace's probing approach can be very noisy. This is supported by the high variance observed for those data points. Vivace converges to fairer allocations for higher bandwidth values. Cubic and BBRv3 both converge to relatively fair allocations when the buffer capacity is set to 0.2× and 1× the BDP. When this is set to 4× the BDP (Figure 4c), both yield less fair allocations. For Cubic, this is because the higher queue size translates to sparser loss events, leading to Cubic flows taking much longer to converge to a fair allocation. BBRv3 converges much slower as queue sizes increase.

3) Fairness in a Multi-Bottleneck Topology: In this section, we explore how the different CC schemes behave in a multi-bottleneck setup, specifically in a *parking lot* topology, which RL-based schemes have not encountered during training. More broadly, such multi-bottleneck scenarios bring a certain level of realism that is otherwise abstracted away when experimenting using dumbbell topologies, and it is important to see how all tested CC schemes perform in terms of fairness. We emulate a network with three bottlenecks, and concurrently start three flows that cross one of those bottlenecks each. A flow that crosses all bottlenecks is initiated 500 RTTs in the experiment. Bottleneck bandwidth is set to 100Mbps and all flows experience the same base RTT. We plot the average ratio of the goodput of the multi-bottleneck flow over the highest goodput achieved amongst the single-bottleneck flows. We look at the final 500 RTTs of each run. In Figure 5, we plot ratios for all tested base RTT values (shown on the x-axis), for different buffer capacities. We also plot lines for max-min and proportional fairness [51].

When the buffer capacity is 0.2× the path's BDP (Figure 5a), Orca stays below proportional fairness; i.e., the multi-

[3]The proof document cited in [20] is unfortunately inaccessible and there is no inter-RTT experimentation in the paper.



(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

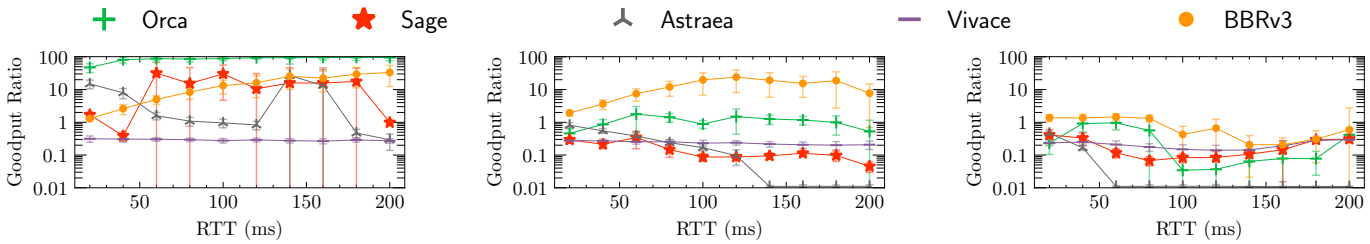
Fig. 5: **Fairness in a Parking Lot Topology**. Bottleneck capacity is 100Mbps, all 4 flows experience the same base RTT (shown on x-axis), buffer capacity is set to 0.2× (a), 1× (b), and 4× (c) the BDP.

bottleneck flow is dominated by the other three flows. This also holds (but is less prominent) when the buffer capacity is 1× the BDP (Figure 5a). When the buffer capacity is 4× the BDP (Figure 5c), Orca comes close to max-min fairness (i.e., all flows get very similar shares of the available bandwidth) for small base RTTs, and close to proportional fairness (i.e., 25 Mbps for the multi-bottleneck flow and 75 for each of the single-bottleneck flows) for larger base RTTs. Sage's fairness is closer to max-min fairness as opposed to proportional fairness. In Figures 5b and 5c, the multi-bottleneck Sage flow is very aggressive, particularly for the first few base RTT values (20, 40 and 60 ms) where its is completely unfair to the single-bottleneck flows (with a ratio that exceeds max-min fairness). When the buffer capacity is small and outside Sage's training range (Figure 5a), fairness varies considerably between different base RTT values, indicating poor generalisation. Astraea is the only learning-based CC scheme that stays close to proportional fairness for all buffer capacities (see Figure 5), when operating within its training parameters. Astraea explicitly accounts for fairness in its reward function, and this is effective in the multi-bottleneck setup despite the fact that one flow competes for bandwidth separately with three other flows. Vivace does not perform well, and for all different buffer capacity and base RTT values, single-bottleneck flows dominate the multi-bottleneck one. This is because the multi-bottleneck flow, which starts after all single-bottleneck flows do, cannot capture enough bandwidth to come close to proportional fairness. The added latency from crossing three bottleneck buffers results in a behaviour similar to the *Inter-RTT Fairness* case above, despite the fact that all flows experience the same base RTT. Cubic's fairness profile is close to proportional fairness; this has been previously observed for AIMD CC schemes [51]. BBRv3 also comes close to proportional fairness for all tested buffer capacity values.

B. Backward Compatibility

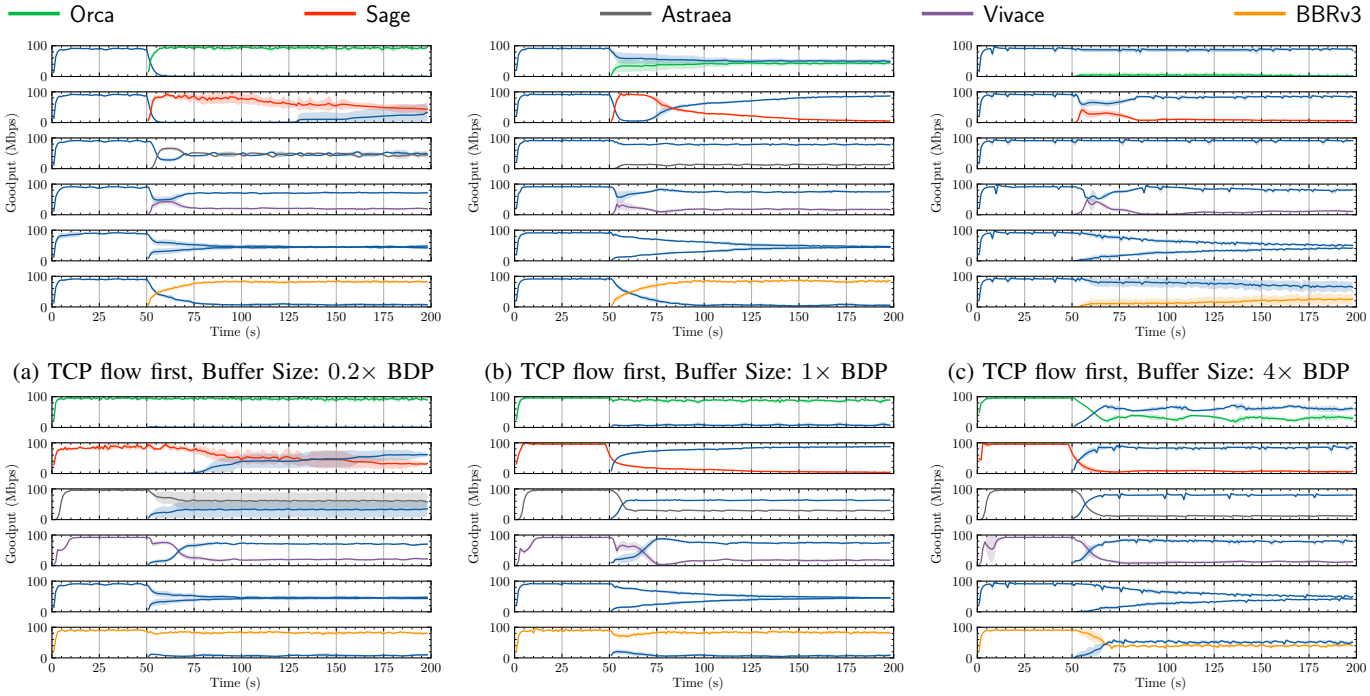
1) Two-Flow Setup: To evaluate the TCP friendliness of learning-based CC, we repeat the fairness experiments but have the first flow running Cubic and the second flow running one of the selected CC algorithms. Figure 6 shows the goodput ratio of the tested flow over that of the Cubic one during the last 500 RTTs of the experiment. A scheme is deemed to be friendly to Cubic as long as it does not take up more bandwidth than Cubic, so an allocation can be unfair but TCP friendly.

When the buffer capacity is set to 0.2× the BDP, Orca is extremely unfair (and unfriendly) when competing with Cubic, with Orca flows dominating Cubic ones, for all base RTT values (Figure 6a). When the buffer capacity is 1× the BDP (Figure 6b), Orca is friendlier to Cubic. Finally, when the buffer capacity is 4× the BDP, Orca is friendly to Cubic but the overall allocation is unfair most of the time (Figure 6c). We provide a plausible explanation about this behaviour below when we look at flow dynamics. To address TCP friendliness, Sage employs an additional reward component that measures how fairly each scheme in its training pool shares bandwidth with Cubic [28]. To showcase its effectiveness, the authors of Sage performed a two-flow experiment under a single parameter instance showing that Sage can fairly share bandwidth with Cubic; we were able to reproduce this specific result. However, in our broader and more systematic experimentation, Sage is shown to achieve an unfair, yet friendly allocation when competing with Cubic, when the buffer capacity is within Sage's training parameter space, where Sage yields most of the bandwidth to the Cubic flow (Figures 6b and 6c). When Sage operates outside its training parameters, it captures more bandwidth than Cubic, with significant variance present in the different runs (Figure 6a). Astraea is generally friendly to Cubic, yielding more and more bandwidth to it, as the buffer capacity increases, eventually leading to completely unfair bandwidth allocations (e.g., above 140ms in Figure 6b and above 60ms in Figure 6c), where Cubic completely dominates the bottleneck. This is because Astraea's reward function penalises latency and therefore the policy will decrease the congestion window as the Cubic flow fills up the buffer. This delay-induced penalty is much lower (to negligible) when the underlying buffer capacity is very small (Figure 6a); as a result, Astraea is friendlier for most of the tested base RTT values, when the buffer capacity is 0.2× the path's BDP with the overall allocation being more fair, too. Vivace is friendly to Cubic when the across the different buffer capacities, although the overall allocation remains unfair, with Cubic taking substantially more bandwidth (Figures 6). With just one Cubic flow competing for bandwidth, the frequency at which the queue gets full is low; therefore, Vivace finds it can reduce RTT and increase its utility by reducing its rate often [20]; consequently, it achieves lower throughput than Cubic. BBRv3 friendliness to Cubic depends on the buffer capacity. With smaller buffer



(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 6: **TCP Friendliness**. Goodput ratio for two competing flows (one being Cubic) in a dumbbell topology. Bottleneck capacity is 100Mbps, flows experience the same base RTT (x-axis), buffer capacity is 0.2× (a), 1× (b), and 4× (c) the BDP.



(a) TCP flow first, Buffer Size: 0.2× BDP (b) TCP flow first, Buffer Size: 1× BDP (c) TCP flow first, Buffer Size: 4× BDP

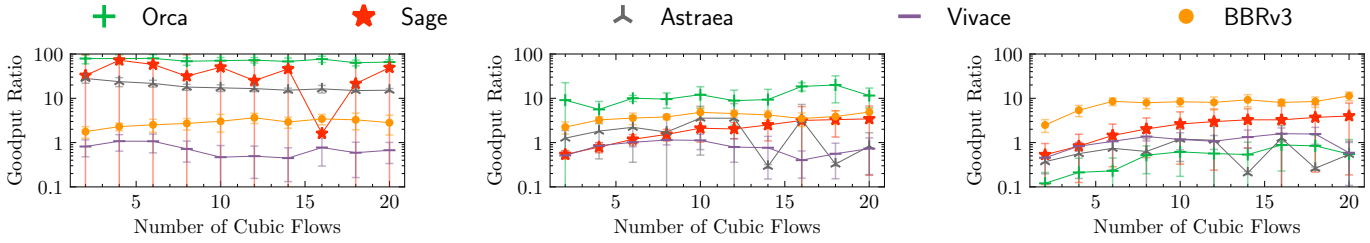
(d) TCP flow last, Buffer Size: 0.2× BDP (e) TCP flow last, Buffer Size: 1× BDP (f) TCP flow last, Buffer Size: 4× BDP

Fig. 7: **TCP Friendliness**. Goodput evolution for two competing flows (one being Cubic) in a dumbbell topology. Bottleneck capacity is 100Mbps, both flows experience the same base RTT (100ms), buffer capacity is set to 0.2×, 1×, and 4× the BDP.

sizes (Figures 6a and 6b), BBRv3 dominates Cubic. When the buffer capacity is set to 4× the BDP and the base RTT values are high (Figure 6c), Cubic fills the buffer, preventing BBRv3 from accurately estimating the bottleneck bandwidth and minimum RTT. As a result, the allocation is friendlier (and fairer) to TCP. It is worth noting that BBRv3 is fairer as the base RTT values get smaller. This is due to the wall clock-based probing mechanism that BBRv2 first introduced to be friendly to loss-based schemes in a parameter space of a typical broadband connection [47].

Flow Dynamics. To better understand flow dynamics when competing with Cubic, we pick a single base RTT value (100ms) and repeat the experiment above in two variations, where Cubic starts first and last, respectively. In Figure 7, we plot the mean and standard deviation (shaded area) of the measured goodput as the two flows compete for bandwidth.

When the buffer capacity is set to 0.2× the BDP of the path, we observe that Orca dominates the bottleneck link, even when the Cubic flow starts first (Figures 7a and 7d); the buffer is so shallow that Cubic will face loss often, allowing the more aggressive Orca flow to dominate the bottleneck. When the buffer capacity is set to 1× the BDP of the path, Orca behaves differently when the Cubic flow starts first compared to when it starts second. In the former case (Figures 7b), the bottleneck link runs at its full capacity when the Orca flow starts, and therefore Orca cannot observe the actual base RTT and maximum bandwidth values, which are crucial for Orca's decision making process. As a result, Orca is less aggressive and friendlier to Cubic, with the allocation converging to a fair equilibrium. When Orca starts first and the buffer capacity is set to 1× the BDP, it completely dominates Cubic (Figure 7e); having observed the base RTT and maximum bandwidth, Orca does not yield much bandwidth to Cubic, which backs off to its aggressive competitor. When the buffer capacity is



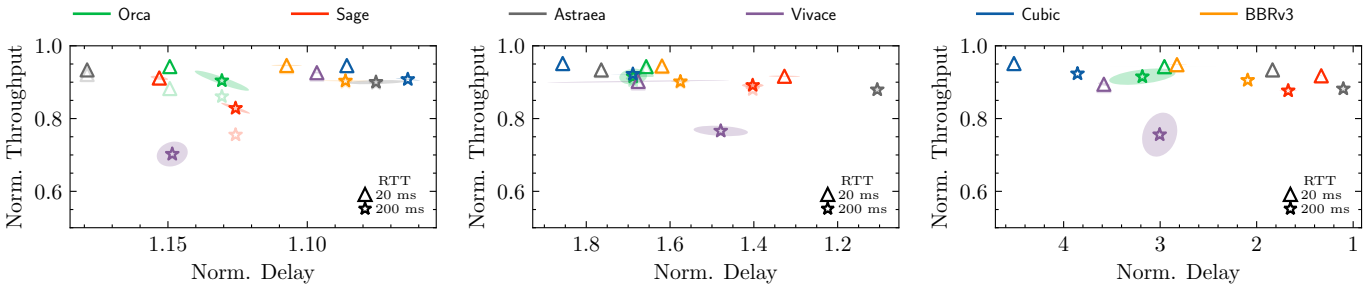
(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 8: **TCP Friendliness**. Goodput ratio between the tested scheme and the average goodput achieved by the Cubic flows. Bottleneck capacity is 100Mbps, RTT is set to 30ms, buffer capacity is set to 0.2× (a), 1× (b), and 4× (c) the BDP. The number of joining Cubic flows is shown on the y-axis.

set to 4× the BDP and the Orca flow starts second, it cannot capture any bandwidth at all; as explained above, Cubic has already captured all available bandwidth, and Orca cannot observe the minimum RTT and maximum bandwidth values, consequently operating at the observed low point (Figure 7c). Finally, when the buffer capacity is set to 4× the BDP and the Orca flow starts first, we observe an interesting pattern in the measured goodput (Figure 7f), with Orca and Cubic periodically approaching and diverging from the fair allocation point. We think this is because Orca's hybrid approach relies on Cubic dynamics on a per-acknowledgment basis and, in this case, Cubic manages to capture some bandwidth due to the buffer capacity being high. Sage, trained using traces from schemes with delay-based CC traits, appears to be less aggressive than (and therefore friendly to) Cubic for all buffer capacity values tested, regardless of the order in which the two competing flows start. When the buffer capacity is set to 1× and 4× the BDP, we observe that Sage eventually gives up all its bandwidth to Cubic (Figures 7b, 7c, 7e, and 7f). Interestingly, this happens even when Sage appears to capture most of the bandwidth during its start-up phase (e.g., as in Figure 7b). When the buffer capacity is 0.2× the BDP, Sage behaves in the same way, although at a much slower pace (Figures 7a and 7d); we can confirm that Sage yields most of its bandwidth, in both cases, when the experiment is left to run for longer (not shown in this Figure). Astraea's reward function is latency-aware and the underlying agent has learned to reduce the congestion window when this would reduce latency. Therefore, when the buffer capacity is large (1× and 4× the BDP), Cubic attempts to fill the available buffer and the consequent latency increase deters Astraea from pursuing further bandwidth. Therefore, Cubic dominates the bottleneck, resulting in a friendly yet unfair allocation; this is the case both when the Cubic flow starts first (Figures 7b and 7c) or last (Figures 7e and 7f). When the buffer capacity is 0.2× the BDP, the latency increase when the buffer gets full is not substantial enough for Astraea to reduce its congestion window. Therefore, it behaves more like a loss-based CC scheme that is friendly to Cubic with the overall allocation being fairer, too. Vivace's behaviour does not depend on which flow starts first. As discussed above, Vivace finds it can reduce its observed RTT and, therefore, increase its utility by reducing its rate often [20]; consequently, it is dominated by Cubic. For this particular RTT, BBRv3 is extremely unfair (and unfriendly) to Cubic when the buffer capacity is 0.2× and 1× the BDP. For smaller buffer capacities, BBRv3 is completely unfair to Cubic (Figures 7a, 7b, 7d, 7e). When the buffer capacity is 4× the BDP mark, BBRv3 converges to a fair allocation; at different paces depending on which flow starts first. This confirms the findings in [39].

2) Multi-Flow Setup: Here, we explore friendliness when a single flow is competing with an increasing number of Cubic flows. Previously, both Astraea and Vivace have been shown to be TCP friendly when competing with many Cubic flows [26] [20], and so we are interested in reproducing these findings and observing how other schemes perform in the same setup. We start the flow of the selected CC scheme first; then, for the next 500 RTTs, we start Cubic flows with their start times selected uniformly at random within this time period. The base RTT and bottleneck bandwidth are set to 30ms and 100Mbps, respectively. In Figure 8, we plot the goodput ratio of the tested flow over the average goodput of the Cubic flows; the number of Cubic flows that we start is shown on the x-axis.

Orca's fairness profile when multiple Cubic flows compete with it is similar to the two-flow setup. For example, in Figure 8a, the Orca achieves a ratio close to 100, which means that the Orca flow completely starves the incoming Cubic flows, which is what we observed in Figure 6a. In Figure 8b we see that Orca yields more bandwidth compared to that in the previous experiment. Similarly, Orca is the friendliest to Cubic when the queue size is 4× BDP (Figure 8c), indicated by the ratio being less than 1. Sage's friendliness to Cubic when the buffer capacity is 0.2× BDP is poor and extremely variable (Figure 8a), with goodput ratio values ranging from 1 to 100. As mentioned in Section IV-A, this value of the buffer capacity is outside the parameter range used in training Sage. When operating within its training range (Figures 8b and 8c), Sage shares bandwidth with Cubic more fairly and with much lower variance, but as the number of Cubic flows grows, Sage becomes more unfair. Interestingly, these results are very much in line with those presented in [28], where the experimental setup involved 3 and 7 Cubic flows competing with 1 Sage flow and it was apparent that Sage becomes more aggressive to Cubic the more Cubic flows are introduced. Although there was no explanation for this behaviour in [28], here we conjecture that Sage, having been trained only with



(a) Buffer Size: 0.2× BDP (b) Buffer Size: 1× BDP (c) Buffer Size: 4× BDP

Fig. 9: **Efficiency**. Aggregate network throughput and average latency for four competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps, flows experience the same base RTT, buffer capacity is set to 0.2×, 1× and 4× the BDP.

setups, including 2 Cubic flows, it has learnt to be more aggressive the more pressure it gets from its competitor. When the buffer capacity is 0.2× the BDP, Astraea maintains its aggressiveness to Cubic, as observed in the two-flow setup, regardless of the number of Cubic flows; Astraea gets at least 10× the goodput of Cubic flows. When we increase the queue size to 1× the BDP, Astraea is friendlier to Cubic flows (Figure 8b), which confirms the results shown in [26]. When the buffer capacity is further increased to 4× the BDP (Figure 8c), Astraea becomes less tolerant to the delay inflation caused by the Cubic flows filling the large buffer, and consequently, yields more bandwidth; the resultant allocation is still friendly but less fair than when the buffer capacity is 1× the BDP. Vivace is shown to be friendly to Cubic across the parameter space explored in this section (i.e., buffer capacity and number of competing Cubic flows). This is in line with the trend[4] presented in [20], further strengthening the authors' intuition related to the reduced utility gain from the RTT gradient, as also discussed above. BBRv3 generally captures more bandwidth than the average Cubic flow, consistent with earlier observations. Specifically, with buffer sizes of 0.2× and 1× the BDP, it captures about 2–3 times more bandwidth than an average Cubic flow. When the buffer capacity is increased to 4× the BDP, the unfriendliness is exacerbated and BBRv3 captures bandwidth more aggressively, due to the BBRv3 flow experiencing a much higher RTT due to the larger buffer being filled by the Cubic flows.

C. *Efficiency*

In this section, we focus on how efficiently the different CC approaches utilise the underlying network resources. We use the same experimental setup as in Section IV-A1, but with four competing flows and experiment with two base RTT values (20ms and 200ms). Flows are scheduled in 25-second intervals, each lasting 100 seconds. We measure aggregate goodput normalised by the capacity of the bottleneck, and average flow latency normalised by the path's base RTT. We also measure the retransmission rate as this affects network utilisation. In Figure 9, we plot mean values and 1-σ ellipses for the 25 seconds all flows co-exist.

[4]A direct comparison is not possible as the experimental setup in [20] uses low bandwidth and allocates buffer proportionally to the number of flows.

When the base RTT is set to 20ms, all CC schemes appear to perform well with flows capturing most of the available bandwidth. Exception to this is Orca, which performs a bit worse when the buffer capacity is set to 0.2× the BDP. We have observed a large amount of retransmissions from Orca flows in this case, which we attribute to the shallow buffer and the highly aggressive sending rate incurred by Orca. Astraea also incurs retransmissions, but not as significantly as Orca does. When the buffer capacity is set to 200ms, the situation is slightly different because the 4 competing flows can be much slower to converge. Consequently, all CC schemes perform worse compared to the 20ms setup; i.e., for each CC scheme, its star marker is below the triangle one. Vivace is affected the most, as shown in Figure 9, particularly when the buffer capacity is 0.2× the BDP. As shown in Section IV-D, convergence for Vivace is very noisy with periods where flows severely under-perform. Previously, in Figure 2c, we observed extensive periods of inactivity for Vivace with just 1 flow running through a bottleneck with a base RTT value of 80ms. Here, we have 4 flows and the base RTT is 200ms and therefore these periods of inactivity because of the reversal in the direction of Vivace's sending rate are much longer. We have confirmed this by looking at individual runs that we do not include here for brevity. Sage's goodput when the capacity is set to 0.2× the BDP is affected by a substantial amount of retransmissions and we attribute that to the fact that it operates outside its training range.

For the latency dimension, we focus on the larger buffer capacity values (i.e., 1× and 4× the BDP) where latency inflation is more apparent and problematic. Orca performs the worst between the RL-based CC approaches, despite including a latency-sensitive component in its reward function. We have previously observed Orca being aggressive in its sending rate and resistant to non-congestive loss, which we believe are the reasons for the latency inflation, despite latency being penalised by its reward function. Sage performs the best in both experimental setups, inflating latency for its flows the least. This is because it has been trained with traces from several latency-aware CC schemes. Astraea also integrates latency in its reward and, in contrast to Orca, this appears to be very effective in keeping latency inflation low, as shown in Figures 9b and 9c. This is the case for both tested RTT values but as we show in the next section Astraea suffers from

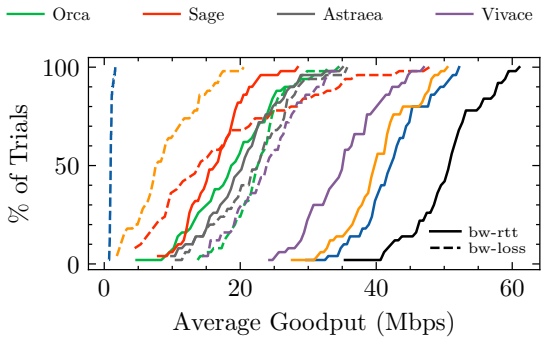


Fig. 10: **Responsiveness**: Cumulative Distribution of Goodput

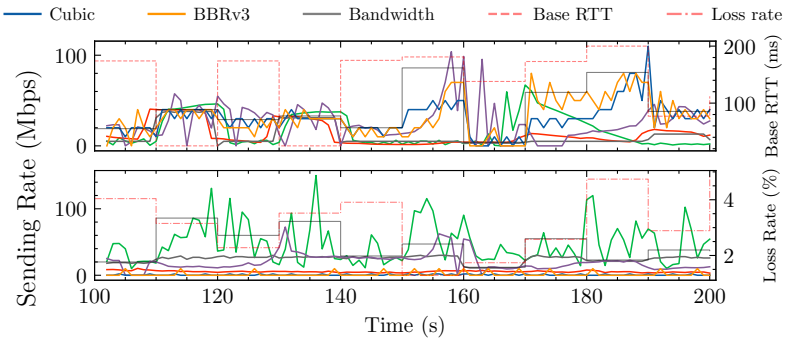


Fig. 11: **Responsiveness**: Sending Rate in Time

substantial unfairness in the 200ms setup.

Cubic and BBRv3 incur significant latency inflation for their flows. Cubic just fills up the buffer that is given, as it is a purely loss-based protocol. In [40], it has been shown that all BBR versions tend to overestimate the bottleneck bandwidth, leading to persistent queue build-up and inflated RTTs. BBRv3 has not introduced mechanisms to explicitly address this issue and is therefore exhibits the same behaviour in this experiment.

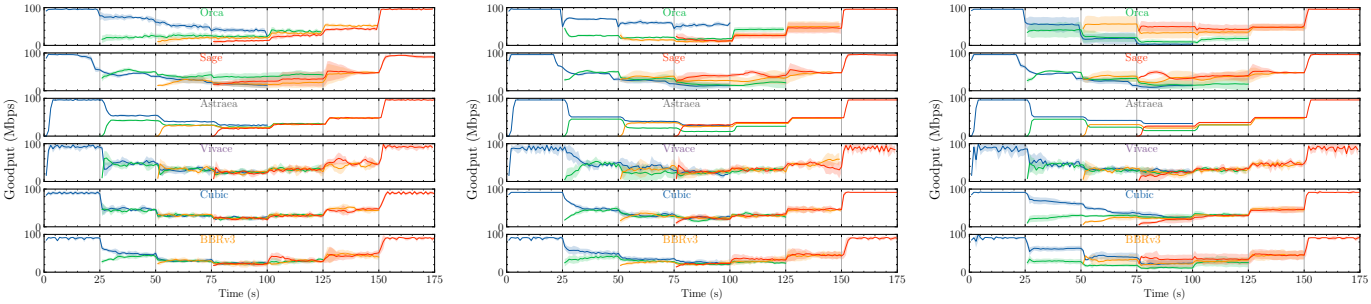
D. Responsiveness

Here, we study how the selected CC approaches react to dynamically changing network conditions. Such changes may be occurring because the underlying network changes (e.g., due to link re-configurations and routing changes, or because the topology itself constantly changes, as in low-earth orbit satellite networks) or because of transient hotspots. We examine how CC approaches behave in the presence of bandwidth and RTT changes, and, separately, bandwidth and non-congestive loss changes.

Varying Bandwidth and Base RTT. We experiment with a single 300-second flow in our emulated dumbbell network. Every 10 seconds we update the bottleneck bandwidth and the base RTT parameters by uniformly selecting values from the ranges 1 - 100Mbps and 20 - 200ms, respectively. For each run, bandwidth and base RTT parameters are the same for all tested protocols. The buffer capacity is set to 638KB (1× the mean BDP). For each CC approach, we repeat the experiment 50 times and calculate the cumulative distribution of average goodput, as in [20]. In Figure 10, the results of this experiment are shown in solid lines. In Figure 10, the black solid line denotes the optimal bandwidth.

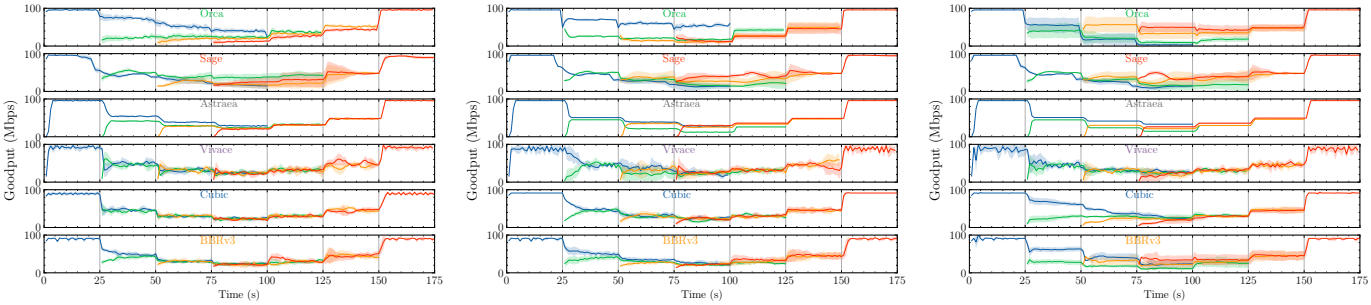
Orca performs badly, particularly when there are jumps in the base RTT. We attribute this to Orca's reliance on estimating the minimum RTT and maximum bandwidth which, in these cases, becomes very problematic. This can be clearly seen in Figure 11, at 140 seconds, where Orca's sending rate collapses when the base RTT increases substantially. Similarly, when the available bandwidth increases, as in Figure 11 at 150 seconds, Orca remains completely unresponsive to the change. Sage performs even worse, frequently not responding to changes in bandwidth and base RTT (as in Figure 11 at 140 and 150 seconds). The traces that Sage uses for training include changing bandwidth values in its step scenarios [28], however, these changes are based on a set of fixed multipliers. In an environment where the bandwidth and base RTT are uniformly selected, as in our experiment in Figure 10, Sage is unable to capture the available bandwidth, achieving the lowest average performance between all CC schemes. Astraea, as with the other RL-based CC schemes, is unable to respond to frequent delay and bandwidth fluctuations, achieving less than half the optimal performance (Figure 10). Note that Astraea is not trained in environments where bandwidth and base RTT change during a training episodes. Like Orca, Astraea uses the minimum observed RTT as a baseline and interprets any increase beyond it as a sign of congestion, clearly seen at 120 seconds in Figure 11. In [20], Vivace is shown to outperform all CC schemes it is compared to in a similar responsiveness experiment. In our study, this is not the case, and Vivace performs much worse than Cubic and BBRv3. This is because, in our experimentation, we adopt a much larger delay range ([20ms, 200ms]) than in [20] ([10ms, 100ms]), which results in a higher average delay experienced by the flow. This, in turn, leads to Vivace being less responsive, because (1) its monitoring interval is based on the underlying RTT and (2) Vivace employs a bandwidth growth cap to filter out erroneous values. This phenomenon is clearly shown in Figure 11 at 150 seconds, where Vivace is very slow to explore the available bandwidth when the delay jumps to 200ms. Cubic performs the best with its bandwidth probing heuristics being effective in tracking the available bandwidth well relative to the RTT it experiences. Similarly, BBRv3 tracks the available bandwidth well. The small deficit compared to Cubic can be attributed to its wall clock bandwidth probes, which cause a slightly slower adaptation to bandwidth increases, as it must wait at most 3 seconds to start the probe bandwidth phase to explore newly available bandwidth. The is clearly shown at 140 and 180 seconds, where BBRv3 is slow to adapt to the RTT increase.

Varying Bandwidth and Non-Congestive Loss. We repeat the experiment above, but now update the bottleneck's random loss probability and bandwidth by uniformly selecting a value from the range 0% - 5% and 1Mbps - 100Mbps, respectively (base RTT is set to 100ms). In Figure 10, the results of



(a) RTT: 20ms, Buffer Size: 0.2× BDP (b) RTT: 20ms, Buffer Size: 1× BDP (c) RTT: 20ms, Buffer Size: 4× BDP

Fig. 12: **Convergence**. Goodput evolution for four competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps, flows experience the same base RTT of 20ms, buffer capacity is set to 0.2×, 1× and 4× the BDP.



(a) RTT: 200ms, Buffer Size: 0.2× BDP (b) RTT: 200ms, Buffer Size: 1× BDP (c) RTT: 200ms, Buffer Size: 4× BDP

Fig. 13: **Convergence**. Goodput evolution for four competing flows in a dumbbell topology. Bottleneck capacity is 100Mbps, flows experience the same base RTT of 200ms, buffer capacity is set to 0.2×, 1× and 4× the BDP.

this experiment are shown in dashed lines. Orca, Sage and Astraea are resistant to random loss. This, in combination with the fact that we do not change the underlying RTT in this experiment, means that they all perform slightly better compared to the non-loss setup discussed above. Once again, in [20], Vivace is shown to perform well with respect to non-congestive loss. However, in our study, we experiment with a much higher loss rate (0 to 5%) than in its evaluation in [20] (0 to 1%), and this is why Vivace performs worse here (still, better than all the RL-based CC schemes). Cubic is known to perform very poorly in the presence of non-congestive loss, being a loss-based protocol; this is clear in Figures 10 and 11 where Cubic's sending rate is suppressed due to the recurring non-congestive loss. BBRv3's sending rate collapses in the presence of substantial non-congestive loss (Figure 11). This is because the average loss rate in this experiment exceeds the 2% loss threshold that BBRv3 uses as a higher bound of bytes in flight [50].

E. Convergence

Figure 12 illustrates convergence for the selected CC approaches; the experimental setup is the one used to measure efficiency (Section IV-C). Orca converges to (relatively) stable goodput values quickly, for all buffer capacity values, but it is obvious that the bandwidth allocation is unfair, as also pointed out in Section IV-A. Similarly, Sage's convergence is swift even when the buffer capacity is high (Figure 12c), but the stable bandwidth allocation is not always fair, with disturbances occurring when new flows join the network. Astraea's convergence is remarkably swift and stable, having little to no variance among the different runs for each different setup. However, not all flows converge to a fair allocation. This contradicts the results shown in [26], where fast convergence and extremely fair bandwidth allocation was prevalent. Although we cannot be certain why this is the case, we believe that this was the result of applying fair queueing in the bottleneck as indicated in the configuration files made publicly available by the authors. Vivace shows a noisy and unstable convergence profile (albeit more fair than Orca and Sage) for all tested buffer capacity values (Figure 12); similar findings were shown in [26]. Cubic flows converge to a fair share when the buffer capacity is 0.2× and 1× the BDP within the 25-second interval, before a new flow enters the network. When the buffer capacity is 4× the BDP, more time is needed for each flow to converge, as shown Figure 12c. BBRv3 convergence is much affected by the buffer capacity, as discussed in Section IV-A1; as the capacity increases, convergence gets slower and less fair.

In Figure 13, we plot the results of the same experiment but with the base RTT values set to 200ms. This value is outside the training parameters of Astraea and Sage. Orca, as with the 20ms case, is very unfair, with some of the competing flows getting close to zero bandwidth. Sage performs better despite the fact that it operates outside its training range; interestingly, one can see that goodput patterns are quite similar to Cubic (for all buffer capacities). This is to be expected because Cubic

is one of the schemes used to train Sage. Astraea is, again, quick to converge to stable bandwidth allocations, but these are very unfair, with the flow started first being very aggressive to all subsequently joining flows. The performance of Cubic and BBRv3 is as expected, with even longer convergence times compared to the 20ms case. Figures 12c and 13c show very similar patterns because the BDP values in the respective experiments are very close to each other.

V. Related Work

Table I shows RL-based CC approaches proposed to date.

Using existing CC schemes in training. Recent learning-based congestion control approaches can be broadly classified into hybrid and clean-slate designs, both of which have evolved since our previous study. Hybrid approaches, those that incorporate existing heuristic CC algorithms during training or operation, have attracted increasing attention. The authors of [28], make the point that although existing schemes are not effective in all scenarios, they nonetheless embody valuable heuristics that can inform learning-based approaches. In the same spirit, Mutant [52], extends this idea by using an online RL agent that continuously switches between existing CC algorithms, selecting the best performer for the observed network context. It formulates selection as a contextual N-armed bandit over a top-k pool chosen offline, then performs state-preserving "mutations" (the state of the current CC schemes is frozen and a new, better performing scheme is selected) when the estimated bandit reward favours another scheme. Another recent example of an RL-based congestion control scheme that leverages heuristic control is ORC [53]. ORC is an online reinforcement learning scheme that operates in conjunction with BBR to prevent the RL agent from making online exploratory decisions that could regress performance relative to the baseline BBR controller.

Towards generalisability. Clean-slate approaches have also made advances, a key contribution being Jury [54]. The aim of Jury is to achieve fairness across a much wider parameter range, motivated by the limitations of Astraea. Jury deliberately removes bandwidth/throughput features from the RL state. Its model is much simpler than that of previous works, with the only inputs to the RL model being changes in normalised RTT and loss. The model outputs a decision range rather than a concrete rate change. In a subsequent post-processing phase, a rate adjustment is made within the decision range given by the model, using the bandwidth occupancy ratio, a metric that can infer how much of the bottleneck a flow is occupying [54]. Evaluation shows that it is effective far beyond the training parameter range seen during training, compared to the state-of-the-art Astraea.

VI. Discussion

In this paper, we reproduced the first systematic large-scale study of RL-based CC approaches, extending it with two more state-of-the-art RL schemes. We have made our experimentation codebase and extracted datasets publicly available to ensure verifiability of our results and promote transparent experimentation. In the following, we discuss key findings and raise the issue of openness in RL-driven research in CC, and hope that our insights will contribute in steering the community to the right directions.

Approach	Implementation	Open Sourced
Aurora [24]	UDT	Yes
DRL-CC [55], Jury [54]	Linux Kernel	No, Not to date
Orca [25], Astraea [26], Sage [28]	Linux Kernel	Yes, Yes, Yes
Spine [27], Mutant [52]	Linux Kernel	No, Yes
Eagle [41], Pareto [42]	C++/Python	No, No
MOCC [56], DeepCC [57]	UDT+CCP	No, Yes
ORC [53]	Userspace (Unclear)	No

TABLE I: RL-based CC schemes implemented in emulation environments

Absence of fairness from reward function. At the time of writing our previous study, fairness was a key limitation of RL schemes, as none explicitly implemented fairness as part of their reward functions. Previous hybrid approaches like Orca that maintains coarse-grain control of an existing scheme (Cubic), showed to be ineffective in regards to fairness. This has since changed with the introduction of Astraea, being the first of its kind to embed fairness directly in its reward function combined with a multi-agent actor critic training framework. As we showed in Section IV-A, Astraea delivers consistently high fairness across different intra and inter-RTT as well as bandwidth values.

Unresponsiveness to dynamic networks. Our results show that existing RL-based CC schemes struggle when the RTT fluctuates frequently, failing to adapt their sending rate to new conditions. The reward formulations of Astraea and Orca rely on minimum RTT or maximum bandwidth estimates, which can become outdated under changing conditions and mislead the policy. In addition, the traces used in training Sage included only fixed-step changes in bandwidth and no changes in RTT, which limit its ability to generalise to the random and frequent fluctuations present in our synthetic experiments and LEO satellite networks, where RTT fluctuations are characteristic. Achieving responsiveness therefore requires a careful balance between training environments and a reward formulation that does not rely solely on the minimum observed RTT.

Reproducibility of Orca. As part of this work, we reproduced the first systematic large-scale study of RL CC schemes, which included Orca. Our results are consistent with the original findings. The only deviation occurred in a single experiment shown in Figure 1a, where we observed higher fairness than reported in [34]. In all other instances, Orca's performance profile remained similar across different hardware configurations, including HyperV virtual machines.

Generalisation of performance to unseen network conditions. Our study shows that RL-based congestion control schemes still struggle to generalise performance to unseen network conditions, for instance (i) Orca struggles to explore bandwidth values even slightly beyond its training range, (ii) Astraea demonstrates strong generalisation in terms of bandwidth and RTT, operating effectively far outside its training parameters, but its fairness properties do not carry over in

such scenarios, (iii) Sage manages to function across a wider space, but its behaviour remains noisy and unstable. These findings highlight that, although RL schemes are effective within their domains, their generalisation remains narrow and highly dependent on the design of training regimes and reward functions.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, 2019.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, 2017.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, 2018.

[5] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[6] B. D. Tracey, A. Michi, Y. Chervonyi, I. Davies, C. Paduraru, N. Lazic *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, 2022.

[7] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, 2019.

[8] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "QARC: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proc. of ACM MM*, 2018.

[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. of ACM SIGCOMM*, 2017.

[10] S. Wang, H. Liu, P. H. Gomes *et al.*, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, 2018.

[11] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, 2018.

[12] T. T. Nguyen *et al.*, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[13] A. Uprety and D. B. Rawat, "Reinforcement learning for IoT security: A comprehensive survey," *IEEE Internet of Things Journal*, 2020.

[14] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Network*, 2018.

[15] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, 2017.

[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, 2008.

[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, 2016.

[18] K. Winstein and H. Balakrishnan, "Tcp ex machina: computer-generated congestion control," ser. SIGCOMM '13. Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2486001.2486020

[19] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: re-architecting congestion control for consistent high performance," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. USENIX Association, 2015.

[20] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-learning congestion control," in *Proc. of USENIX NSDI*, 2018.

[21] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental evaluation of TCP protocols for high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, 2007.

[22] K. Xiao, S. Mao, and J. K. Tugnait, "TCP-Drinc: Smart congestion control based on deep reinforcement learning," *IEEE Access*, 2019.

[23] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, 2019.

[24] N. Jay, N. Rotman, B. Godfrey *et al.*, "A deep reinforcement learning perspective on Internet congestion control," in *Proc. of ICML*, 2019.

[25] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the Internet," in *Proc. of ACM SIGCOMM*, 2020.

[26] X. Liao, H. Tian, C. Zeng, X. Wan, and K. Chen, "Astraea: Towards fair and efficient learning-based congestion control," in *Proc. of EuroSys 2024*, 2024.

[27] H. Tian, X. Liao, C. Zeng *et al.*, "Spine: an efficient DRL-based congestion control with ultra-low overhead," in *Proc. of ACM CoNEXT*, 2022.

[28] C.-Y. Yen, S. Abbasloo, and H. J. Chao, "Computers can learn from the heuristic designs and master internet congestion control," in *Proc. of ACM SIGCOMM*, 2023.

[29] N. Handigol, B. Heller *et al.*, "Reproducible network experiments using container-based emulation," in *Proc. of CoNEXT*, 2012.

[30] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "QTCP: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, 2018.

[31] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *Proc. of USENIX NSDI*, 2020.

[32] L. Giacomoni, B. Benny, and G. Parisis, "RayNet: A simulation platform for developing reinforcement learning-driven network protocols," *CoRR*, vol. abs/2302.04519, 2023.

[33] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI Gym: The playground for machine learning in networking research," in *ACM MSWiM*, 2019.

[34] L. Giacomoni and G. Parisis, "Reinforcement learning-based congestion control: A systematic evaluation of fairness, efficiency and responsiveness," in *Proc. of IEEE INFOCOM*, 2024.

[35] J. P. Hespanha, S. Bohacek, K. Obraczka, and J. Lee, "Hybrid modeling of TCP congestion control," in *Proc. of HSCC*, 2001.

[36] R. Shorten, C. King, F. Wirth, and D. Leith, "Modelling TCP congestion control dynamics in drop-tail environments," *Automatica*, 2007.

[37] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *Proc. of IFIP Networking*, 2018.

[38] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, 2007.

[39] D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, "Promises and potential of bbrv3," in *Passive and Active Measurement: 25th International Conference, PAM 2024, Virtual Event, March 11–13, 2024, Proceedings, Part II*, 2024. [Online]. Available: https://doi.org/10.1007/978-3-031-56252-5_12

[40] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "Evaluating tcp bbrv3 performance in wired broadband networks," *Computer Communications*, vol. 222, pp. 198–208, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366424001658

[41] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *Proc of IEEE INFOCOM*, 2020.

[42] S. Emara, F. Wang, B. Li, and T. Zeyl, "Pareto: Fair congestion control with online reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, 2022.

[43] "iperf3." [Online]. Available: https://iperf.fr/

[44] S. Hemminger, "TCP testing (tcpprobe)," *The Linux Foundation, http://devresources.linuxfoundation.org/shemminger/tcp*, 2011.

[45] S. Godard, "SYSSTAT home page," *Information and code available at http://sebastien.godard.pagesperso-orange.fr/index.html*, 2015.

[46] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. of ACM HotNets*, 2010.

[47] "BBR3 enabled kernel." [Online]. Available: https://github.com/google/bbr/blob/v3/net/ipv4/tcp_bbr.c#L866

[48] "The dataset contains measurements collected from many experimentals, of various cc schemes including orca, sage, astraea, pcc vivace (kernel and userspace), cubic, and bbrv3 and bbrv1 in emulated dumbbell and parking-lot topologies. this dataset is used to generate the figures." [Online]. Available: https://figshare.com/s/97c09c17972fb0ca56b4

[49] "Code repository for the experiment and plotting scripts." [Online]. Available: https://github.com/Aruuni/mininettestbed

[50] N. C. Y. C. K. Y. D. M. S. H. Y. P. J. Y. Seung, "Bbrv3: Algorithm bug fixes and public internet deployment," 2023. [Online]. Available: https://datatracker.ietf.org/meeting/117/materials/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment-00

[51] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing tcp," 1998.

[52] L. Pappone, A. Sacco, and F. Esposito, "Mutant: learning congestion control from existing protocols via online reinforcement learning," ser. NSDI '25. USENIX Association, 2025.

[53] Y. Li, J. Huang, C. Wu, X. Zhu, and J. Wang, "Orc: Online reinforcement learning for congestion control with fast convergence," in *Proceedings of the 9th Asia-Pacific Workshop on Networking*, ser. APNET '25, 2025. [Online]. Available: https://doi.org/10.1145/3735358.3735381

[54] H. Tian, X. Liao, D. Sun, C. Zeng, Y. Jin, J. Zhang, X. Wan, Z. Wang, Y. Wang, and K. Chen, "Achieving fairness generalizability for learning-based congestion control with jury," ser. EuroSys '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3689031.3696065

[55] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path TCP meets deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, 2019.

[56] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," in *Proc. of EuroSys*, 2022.

[57] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna make your tcp scheme great for cellular networks? let machines do it for you!" *IEEE Journal on Selected Areas in Communications*, 2021.