Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control

Chen-Yu Yen[★†], Soheil Abbasloo[★‡]
[★]Equal Contribution

H. Jonathan Chao[†]
[†]New York University and [‡]University of Toronto

ABSTRACT

In this work, for the first time, we demonstrate that computers can automatically learn from observing the heuristic efforts of the last four decades, stand on the shoulders of the existing Internet congestion control (CC) schemes, and discover a better-performing one. To that end, we address many different practical challenges, from how to generalize representation of various existing CC schemes to serious challenges regarding learning from a vast pool of policies in the complex CC domain and introduce Sage. Sage is the first purely data-driven Internet CC design that learns a better scheme by harnessing the existing solutions. We compare Sage's performance with the state-of-the-art CC schemes through extensive evaluations on the Internet and in controlled environments. The results suggests that Sage has learned a better-performing policy. While there are still many unanswered questions, we hope our data-driven framework can pave the way for a more sustainable design strategy.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Public Internet**; **Network servers**; • **Computing methodologies** → **Neural networks**; • **Theory of computation** → **Reinforcement learning**;

KEYWORDS

TCP, Congestion Control, Data-Driven Reinforcement Learning

ACM Reference Format:
Chen-Yu Yen, Soheil Abbasloo, H. Jonathan Chao. 2023. Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3603269.3604838

1 INTRODUCTION

Setting the Context: The task of controlling congestion on the Internet is one of the challenging and active research topics in the network community. The challenging nature of the congestion control design on the Internet comes from some of its essential aspects, including access to imperfect information, large action space, and its distributed cooperative game nature that manifests itself in the form of TCP-friendliness and fairness. Early results during the '80s (e.g., [41]) showed that due to these immense challenges, in practice, reaching the theoretical optimal operation points is not feasible. These results motivated the community toward the use of heuristic designs in the last four decades to control congestion (e.g., [7, 12, 14, 17–20, 28, 30, 34–36, 40, 45, 48, 58, 68][1]).

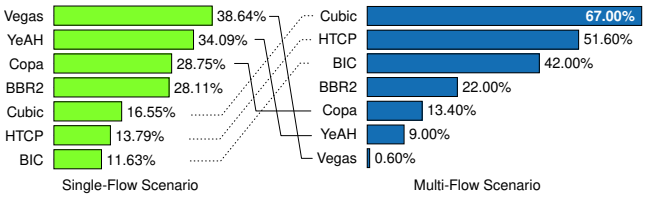


Figure 1: Winning Rates of some CC schemes in scenarios of Set I (left) and Set II (right) defined in Section 5

The Empty Half of the Glass: An important lesson from the decades of CC design is that although these schemes might do a good job in certain scenarios, they fail in other ones. As an illustrative example, following the setup described in more detail in section 5, in a first set (Set I), we focus on the gained throughput and end-to-end delay of some CC schemes in single-flow scenarios, and in a second set (Set II), we examine their TCP-friendliness by capturing their performance when they share the network with TCP Cubic [35] flows (the default CC scheme in most of the platforms). Generally, a policy achieving higher throughput and lower delay in Set I is considered better. In Set II, a policy that can share the link more fairly is regarded as a better-performing policy. Fig. 1 shows the percentage of times that a scheme's performance is either the best or at most 10% less than the best-performing scheme. As expected, none of these schemes can always be the best-performing scheme and interestingly, the schemes' ranking in Set I is quite the opposite of their ranking in Set II.

The Full Half of the Glass: Looking from a different angle, another important fact is that every existing scheme, which embodies a *manually discovered policy* mapping carefully chosen input signals/observations to carefully handcrafted actions, has some design merits and manages to perform well, though only in certain scenarios. That said, the vast pool of already discovered policies embodies an accumulated knowledge gained throughout years of CC research and that is why this pool of policies is precious[2].

The Key Question: The main design approach used in the last decades is to repeat the cycle of trying to *manually* (1) investigate existing schemes, (2) learn from their pros and cons, and (3) discover

[1]Appendix A briefly overviews some of these schemes
[2]This work does not raise any ethical issues.

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09. ·$15.00
https://doi.org/10.1145/3603269.3604838

another heuristic. However, the overabundance of existing schemes combined with the empty half of the glass (the fact that each of these CC schemes fails in some scenarios) sheds light on why this design approach is inadequate, time-consuming, and unsustainable. That is why in this work, we take a different design philosophy aiming for *automatically* harnessing the precious pool of existing policies. In particular, we target answering a refreshing key question that can potentially pave the way for a more sustainable design strategy:

Q1. Can machines learn solely from observing the existing CC schemes, stand on their shoulders, and automatically discover a better-performing policy?

1.1 Main Design Decisions

Why not Behavioral Cloning (BC)? When it comes to learning a task from only observing the collected experiences of some experts, the most straightforward approach is BC [56]. In BC, the goal is to mimic an expert by learning from demonstrations of the expert's actions. Putting aside the well-known algorithmic restrictions and issues of BC (see section 6.2), the main reason BC is not a good fit is although we intend to learn from observing the existing schemes, our key goal is to learn to surpass them, not to be similar to them.

Why not Directly Learning from Oracles? One may try to use imitation learning to directly learn from the *optimal* behavior that perfectly maps states to actions. However, there are different issues with this oracular-based approach in the context of CC. For instance, modeling CC in a general setting is intractable. That means how to find the CC oracles in different environments for itself is a very complicated task. Moreover, even when CC oracles are approximated in some simple settings, as we show later in section 6.2, the final learned model cannot perform well in other scenarios.

Why not (Online) Reinforcement Learning (RL)? RL algorithms are essentially *online* learning paradigms where agents iteratively interact with an environment, collect experience, and use it to improve a policy/behavior. However, in practice, this online aspect of RL brings some severe issues to the table, especially with respect to effective generalization in complex domains [44, 46, 57]. In other words, effectively training an RL agent in complex domains - with high-dimensional, nonlinear parameterizations - is very hard. This becomes one of the biggest obstacles to realizing online RL algorithms. This is also the case in the context of Internet CC. As we show later in section 6.2, the state-of-the-art online RL algorithms experience serious performance issues when utilized for long training sessions over a large set of environments. Moreover, online RL CC schemes are generally clean-slate designs and in contrast to our key goal, cannot cherish the existing pool of heuristic CC designs.

Our Approach: Considering these discussions, to answer *Q1*, for the first time, we design a CC framework based on advanced *data-driven (offline)* RL techniques. In a nutshell, our data-driven RL framework enables exploiting data collected ***once***, ***before*** training, with ***any*** existing policy (section 2 provides background on data-driven RL and its main differences compared with online RL). Using our framework, we show that, indeed, computers can learn and unveil a better-performing CC scheme by solely observing the performances of the existing ones. We refer to finding such a policy as mastering the task of CC and call our design Sage (of CC)

Disclaimer: Note that mastering a task does not necessarily mean achieving the optimal performance. A simple analogy is a chess player who got the title of Master. She is not necessarily the perfect chess player, but she has gained mastery of the game by achieving overall good performance among other players [63]. That said, Sage is not *the* optimal algorithm, but as our experiments (section 6) indicate, overall, it performs better than the existing ones.

1.2 Some Challenges and Contributions

Realizing an effective data-driven framework to not only address *Q1* but also successfully compete with state-of-the-art schemes in practice, faces various practical challenges.

1) Algorithmic Challenges: A clear key challenge is how to design and utilize a system that can generate sufficient variations of various policies, effectively learn from the generated pool of policies, and gain a model that can compete with the state-of-the-art CC schemes in practice.

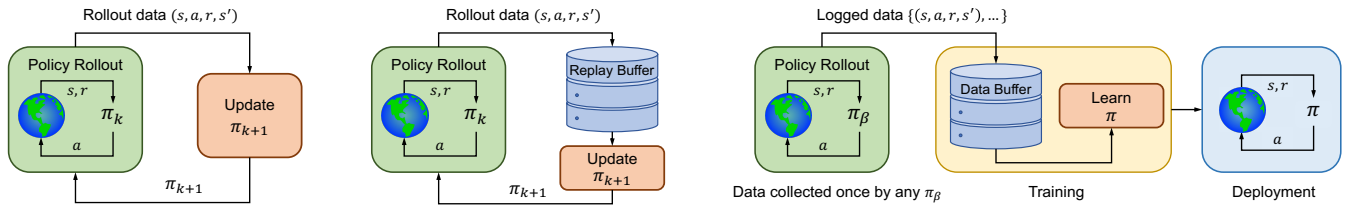
2) General Representation: How the general representation for inputs/outputs of our system should look so that it can cover different existing CC schemes with their intrinsically different requirements, while it is not bound by them?

3) The Myth of "the More Training, the Better Result": In the network community, there is a general impression that any ML-based system (including RL-based ones) always will be improved when trained for *longer* duration and in *wider* settings (more environments). Consequently, if a learning-based design already exists in a certain domain (such as CC), it is assumed that it is just a matter of training it *longer* and *wider* to perform the task. However, as we show in section 6.2, this is a wrong impression and indeed, how to learn in wider and longer settings is one of the crucial obstacles and challenges to realizing effective learning-based algorithms.

4) Training on General-Purpose Clusters: While general-purpose clusters for running ML training tasks are more accessible, the state-of-the-art ML-based CC schemes (e.g., Orca [9]) require a cluster of customized servers (e.g., patched with new Kernel codes or with access to underlying Kernel services for generating packets) to effectively perform their training. Similar requirements greatly complicate the training phase and can prevent people with no luxury of accessing big custom clusters, to have tangible impacts in this domain.

Contributions: Addressing these and similar challenges, our main contributions in this paper are as follows.

- We design Sage (detailed in sections 3 and 4), the first data-driven RL framework that, in contrast with its existing ML-based CC counterparts, can be successfully trained over a large set of networks even without the need for sending a packet in those networks during the training phase (detailed in section 5).
- We extensively evaluated Sage (detailed in section 6) and demonstrated that, indeed, it is feasible to effectively learn from the existing heuristic CC schemes. Our data-driven CC framework presents a non-zero-sum design philosophy cherishing, not alienating, the decades of heuristic designs.
- We made our data-driven framework publicly available to facilitate future contributions of the community to data-driven approaches.



(a) Online On-Policy RL **(b) Online Off-Policy RL** **(c) Data-Driven/Offline RL**

Figure 2: Online RL (a and b) must access env. during training, while data-driven/offline RL (c) works without this access

2 ONLINE VS. DATA-DRIVEN RL; A BRIEF BACKGROUND

Online RL: RL is a framework for sequential decision making. An RL agent aims to optimize user-specified reward functions by actively interacting with the environment in a trial-and-error fashion. The recent introduction of deep neural networks as effective high-capacity function approximators into the RL and the excellent results of this combination in a wide range of domains exponentially have increased the popularity of RL.[3] On the other hand, one of the key reasons for progress in deep learning (DL) is access to more data [33]. However, the fact that RL is intrinsically an online paradigm and requires interaction with the environment to collect data during training remains one of the main obstacles to its widespread adoption, especially in complex domains where effective generalization requires large datasets [44, 46, 57]. Note that the online aspect of RL does not mean that the data must be collected necessarily in a *live* manner. For instance, in a simple game scenario, a reward, when defined in the form of a binary of win or lose, can be collected only after an iteration of the game is played by executing a policy. In other words, regardless of whether collecting a reward occurs during an iteration of a game or after that, the online aspect of the RL points to the fact that the agent should be able to interact with the environment and collect more data *during the training*.

On-Policy and Off-Policy RL: In a more classic online RL setting, the agent uses an *on-policy* fashion to collect data. In an on-policy method, the agent runs a policy π_k, collects the impact of π_k on the environment, and then uses this newly collected data to come up with π_{k+1}. In contrast, in an *off-policy* setting, the agent can update its policy by using not only the currently collected experiences, but also the previously gathered ones. This significantly increases the sample efficiency during the training, because as opposed to the on-policy method, old experiences will not be simply discarded. Off-policy RL maintains a replay buffer that stores collected samples and reuses them for temporal difference learning with experience replay. In the context of CC, for instance, Aurora [42] uses an on-policy algorithm, while Orca [9] exploits an off-policy method that enables it to utilize distributed training. Although off-policy mitigates the data collection issue of online RL, it remains online at its core and still shows severe issues when training spans a large set of environments and long durations (e.g., see section 6.2).

Data-Driven/Offline RL: Data-driven algorithms have tremendous success in different domains such as computer vision [43] and natural language processing [23]. The produced models show remarkable expressivity and general knowledge as they learn from more and more data. Although it is more natural to incorporate data in supervised settings, it presents difficulties in the online RL paradigm [44, 57]. To address that, the novel data-driven/offline RL paradigm is introduced. Simply put, data-driven RL aims to solve the same problem that online RL tries to solve, but without the need to interact with or collect data from the environment during the training. Instead, data-driven RL only exploits data collected ***once*** with ***any*** policy ***before*** training. Fig. 2 depicts differences of data-driven RL and online RL. For a more detailed comparison, readers can check existing surveys/tutorials (e.g., [46] and [57])

[3]For example, in the network community, RL has recently been used in different domains from CC (e.g., [9, 42]) to buffer and network management (e.g., [70, 74]).

3 SYSTEM DESIGN OVERVIEW

This section overviews our data-driven CC framework. Later, in section 4, we elaborate on its components. As Fig. 3 indicates, Sage consists of three main blocks namely: (1) Policy Collector, (2) Core Learning component, and (3) Execution block.

Policy Collector: The primary part of any data-driven system is the access to data. In the CC context, data should indicate the behavior of existing CC policies/schemes in different network scenarios. The Policy Collector block is responsible for generating and maintaining such a pool of policies. As we explain more in section 4.1, instead of simply using a union set of various input/output signals of different existing CC schemes, we exploit a general representation model for the input and output signals. In other words, instead of digging into the details of each scheme and figuring out the exact input signals that specific schemes use in their logic, we consider individual CC schemes as black boxes that receive pure raw input signals and *somehow* map them to general CC actions. So, suppose a particular CC scheme uses an engineered input signal, which is not directly part of our available raw input signals. In that case, we assume that this engineered signal is just another hidden part of the logic of that particular scheme. This enables us to generalize existing policies and learn a potentially better one.

Core Learning Component: The output of the Policy Collector block is a pool of {state, action, reward} records captured by monitoring the behavior of different CC schemes in different environments. Our Core Learning block aims to harness this precious pool and, by utilizing advanced data-driven RL algorithms and techniques, find a policy that can perform better than the existing ones. An important feature of our learning block is that it does not try to mimic the existing policies. On the contrary, as a data-driven RL framework, it observes the pool of {state, action, reward} records

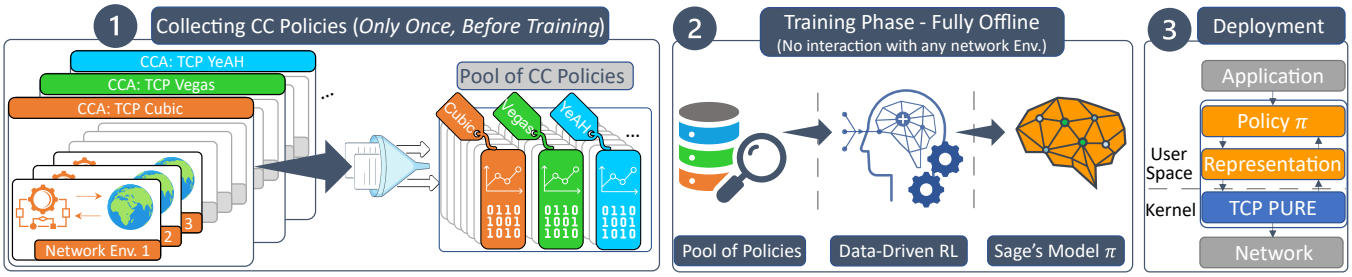


Figure 3: Sage's 3 phases: (1) generating a pool of existing CC policies, (2) data-driven RL training, and (3) deployment

as just a set of trajectories resulting from executing some arbitrary (and not necessarily good) policies. Simply put, it tries not only to learn from the actions of the existing policies that led to good performances but also from the bad actions that led to poor performances. This phase solely relies on the data collected before the training. Therefore, during training, we do not require any new interactions with any network environment. In practice, that gives us a great advantage during the training phase. In particular, we can run the training phase on the commercial general-purpose GPU clusters in which patching the Kernel code or access to underlying Kernel services for sending/receiving packets over custom-built network environments are not usually allowed. We elaborate more on the details of this block in section 4.2.

Execution Block: The output of the learning block is a policy that the Execution block will use. This block represents the deployment phase of Sage. Although a CC policy is the central part of a transport layer, it is not the only part. For example, TCP provides other functionalities either during the connection establishment phase or during the life of a connection. So, to not reinvent the wheel, we use these already existing blocks to complete the design and deployment of Sage. To that end, we design and implement a minimal Kernel module called TCP Pure that inherits the general functionalities of a TCP scheme and provides new APIs to efficiently interact with the gained policy by providing required input signals and enforcing actions received from the policy. For the sake of space, we omit the details of this block and refer readers to Sage's source code [1].

4 SAGE'S DESIGN

4.1 Policy Collector

As Fig. 4 shows, the Policy Collector block collects a pool of CC policies by employing three main components: (1) General Representation (GR) Unit, (2) Kernel-Based CC Schemes, and (3) Network Environment. In a nutshell, a network emulator uses TUN/TAP interfaces to model different scenarios and provide the underlying network environments carrying packets from a source to a destination. The Kernel-based CC Algorithm (CCA) will control the sending rates of the traffic. On top of that, the GR Unit employs Kernel socket APIs and periodically records some general statistics of the traffic and the action of underlying CC schemes. These recorded stats make a pool of policies representing behavior of schemes over different environments.

Input Representation: Any CCA has been designed considering a set of so-called congestion signals. For instance, a loss-based CCA (e.g., Cubic [35] and NewReno [34]) uses the loss of packets as the main input signal to handle congestion, while a delay-based CCA (e.g., Vegas [17]) focuses on the delay of packets as the main congestion signal. In general, congestion signals can even include more complicated statistics such as the first or second derivatives of the delay. That said, one key decision in the design of Sage is that we favor simplicity and generalizability over engineered input signals. There are a few main reasons behind this decision. First, any handcrafted input signal intrinsically emphasizes certain CC strategy, and in the end, increases the chance of learning only strategies similar to those ones. Second, it is reasonable to think that a better CC policy requires more diverse congestion input signals potentially beyond the ones that are already used in the literature. Third, one fundamental motivation for a learning-based CC design is to let the system itself learn/figure out the importance of different input signals. That said, the GR unit collects three general categories of input signals: (1) delay oriented, (2) throughput oriented, and (3) loss oriented signals. For each category, statistics such as average, max, and min are collected. Later, in ablation studies of section 7.3, we show the importance of collecting these statistics.

Another key design decision is centered around the choice of granularity of these statistics. Although Sage's logic performs periodically (in small time intervals), the calculation of these statistics should not necessarily happen with the same time granularity. In particular, as discussed in section 7.4, we found out that indeed larger granularity for the calculation of these stats greatly helps in scenarios where other flows coexist in the network, while medium and smaller granularity can be helpful in single-flow and variable link scenarios. The rationale behind this is that objectives such as TCP-friendliness that involve a long-term goal (such as achieving on average a fair share of bandwidth when competing with other flows) intrinsically depend more on the history of the network than the current state of it. On the other hand, objectives with a more

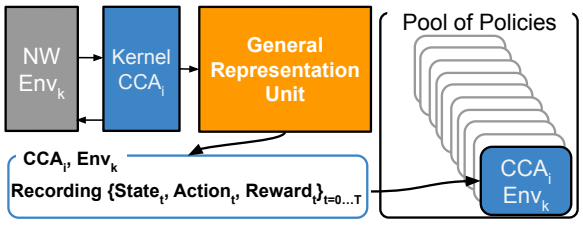


Figure 4: Policy Collector block

myopic nature such as achieving high throughput or low delay for the packets depend more on the current state of the network. So, we use three different timescales to calculate different input signals.

Putting all together, Policy Collector represents the state of the network at timestep t by a vector with 69 elements. Table 1 (Appendix B) gives more details of the input vector.

Output Representation: Generally, a CC scheme decides how many packets should be sent to the network (indicated by a so-called congestion window, *cwnd*). Different CC algorithms apply different sets of actions to adjust their *cwnd*. For example, some classic actions include "$cwnd \mathrel{+}= \frac{1}{cwnd}$", "$cwnd \leftarrow \frac{cwnd}{2}$", and "$cwnd \mathrel{+}= 1$". On the other hand, the actions of a CC scheme can happen at different time scales and magnitudes. That makes the policy collection and learning intractable. To address this challenge, instead of tying the output to how certain CC schemes perform their actions, the GR unit employs a general output representation. In particular, using provided Kernel APIs, the GR unit periodically records the values of *cwnd* and represents the output of the underlying CC scheme at timestep t as $a_t = cwnd_t/cwnd_{t-1}$. The GR unit uses the ratio rather than the exact values of *cwnd* to better generalize to different magnitudes of *cwnd*. In other words, the rationale behind this design choice is that the exact values of *cwnd* depend highly on the exact parameters of the network, while the behavior of relatively increasing/decreasing them has a less direct tie to the exact network setting. That said, we prefer to learn the behavior not to simply and mistakenly memorize the exact values in different settings.

Rewards: The reward is a scalar that quantifies the performance of a CC scheme and it gives Sage a sense of how well different schemes perform in different conditions. Instead of a single reward term (as it's the case for most prior work), we consider two separate reward functions targeting two different design goals simultaneously. The first goal which has a myopic nature centers around optimizing the individual flow's objectives such as high link utilization, low delay, and low loss. Inspired by prior work and the well-studied metric of Power [9], in single-flow scenarios, the GR unit assigns the following reward term at timestep t:

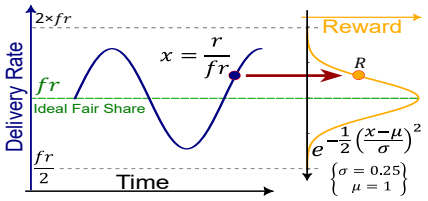
$$R_{1,t} = \frac{(r_t - \xi \times l_t)^{\kappa}}{d_t} \quad (1)$$

where r_t, l_t, and d_t are the delivery rate, loss rate, and average delay observed at timestep t, respectively. Also, ξ and κ determine the impact of the loss rate, and the relative importance of throughput with respect to delay, respectively.

The second design goal, which has a more farsighted nature, is to optimize for the TCP-friendliness. In particular, when a default popular loss-based CC scheme coexists with a particular CC scheme and shares the same bottleneck link, the GR unit aims to quantify how well that particular CC scheme shares the bandwidth with the loss-based scheme[4]. That said, equation 2 formulates this reward where r_t and fr_t are the current delivery rate and the ideal fair share of the bandwidth for that certain CC scheme. The intuition behind $R_{2,t}$ is depicted in Fig. 5.

Environments: To generate different network environments, we use emulation and mainly control four key settings in the network:

[4]In particular, due to fact that TCP cubic is the default CC scheme in most of the platforms, we use Cubic as the default loss-based scheme.



$$R_{2,t} = e^{-8(x_t-1)^2} \quad (2)$$

$$(x_t = r_t/fr_t)$$

Figure 5: TCP-friendliness Reward

the link capacity, minimum end-to-end delay, bottleneck link buffer size, and the existence/non-existence of competing Cubic flows. For any given $\mathbf{Env}_i$, we run the CC scheme j, record the {input, output, reward} vectors throughout time, and finally gain the $\mathbf{Policy}_{ij}$. We make the pool of policies by repeating the procedure for all environments and CC schemes. A very important point here is that the policy collection phase happens only once and after generating the pool, all environments are unplugged. So, during the training phase, no interaction with any environment occurs. Another key point is that Sage can utilize any labeled data and this is not restricted to data collected from emulations. For instance, one can collect data by running different schemes over real networks, though the collected data should be labeled with proper rewards. That said, we use emulation, because it provides a simpler mechanism to fully control the network and have access to reliable rewards. Note that except the network, no other parts are simulated or emulated. This includes generating, sending, and receiving real packets that are based on utilizing the Kernel code and APIs. That way, we are able to capture different real phenomena such as extra delays due to the Kernel, Ack accumulations, etc.

4.2 Core Learning Block

Architecture/Network: The pool of CC schemes forms a broad spectrum of policies. Considering flexibility and generality, we do not assume a predefined handcrafted mechanism in Sage's model. Instead, the model should learn the knowledge directly from the data on its own. However, learning a data-driven policy is challenging due to the need for extracting meaningful features from high-dimensional data, allowing the model's decision to propagate over time, and increasing the stability of training over extensive data. To address these challenges, we design a deep architecture as demonstrated in Fig. 6. Sage uses deep policy networks that take input observation and output the CC action, with the spaces described in section 4.1, to achieve high expressivity.

The input state s goes through the encoder, which processes the high-dimensional raw observations and converts them into feature embeddings. Since some CC heuristics use somewhat memory-based decision-making mechanisms, we use Gated Recurrent Units (GRU) [21] to capture sequence-level information from the data. By using GRU, the agent can propagate its hidden states throughout time. The output goes to another encoder, then a fully-connected (FC) layer followed by two residual blocks. We use residual blocks [37] together with Layernorm to reduce the difficulty of training large models and increase stability [13, 47]. Since our goal is to learn from multiple arbitrary policies, we employ a stochastic action distribution parameterized by a Gaussian mixture model (GMM) [26] in the

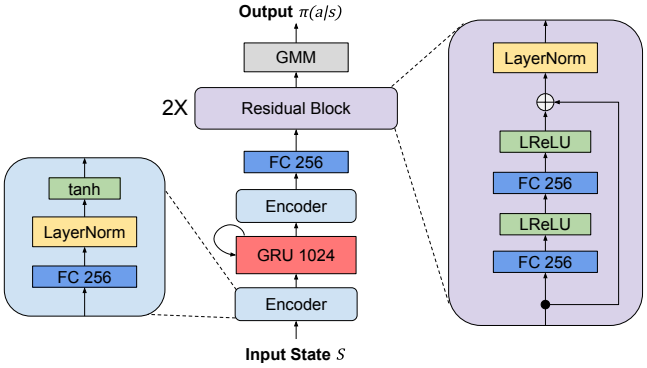


Figure 6: Sage's neural network

last layer of the policy network. GMM helps improve the learning by mitigating the chance of converging to a single arbitrary CC heuristic. Finally, we obtain the output action a_t by sampling from $\pi(a|s)$. Later, in ablation studies of section 7.3, we elaborate more on the importance of these components.

Main Learning Algorithm: The goal of our agent is to use the pool to learn a policy without interaction. Consider a Markov decision process (MDP) $M = (S, \mathcal{A}, P, \gamma, R)$, where S is the state space, $\mathcal{A}$ is the action space, P is the environment dynamics, R is the reward $\{R_1, R_2\}$, and γ is a discount factor. The objective is to find a $\pi(a|s)$ that maximizes the cumulative reward. The Q function describes the expected cumulative reward starting from state s, action a, and acting according to π subsequently:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[\sum_{t}^{T} \gamma^t R_i(s_t, a_t)\right] \quad s_0 = s, a_0 = a, s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t) \tag{3}$$

We let the pool of policies be $\bigcup_{i,j} \textbf{Policy}_{ij}$. A CC behavior policy, j executed in a network environment, i, will generate a trajectory τ, which consists of a sequence of transitions $(s_0, a_0, r_0, ..., s_T, a_T, r_T)$. We store the generated trajectories in the data buffer D. Our objective is to find a policy π that maximizes the expected cumulative reward J using D:

$$J = \mathbb{E}_{(s,a)\sim D}[Q^{\pi}(s, a)] \tag{4}$$

One solution to extract the policy π is to take an *arg* max operation on Eq. 4, when Q and π are parameterized by neural networks. However, simply using *arg* max leads to some well-known issues such as overestimation issue [31]. When the agent tries to select an action far from samples gathered in D, it causes the agent to produce problematic values during the training, and the errors will propagate through the Q function and the policy. Another issue that will appear in the deployment phase of the learned congestion control policy is that a bad *cwnd* could potentially deviate the agent into bad situations leading to unusable behavior. To address these challenges, we build our agent on top of a form of Critic-Regularized Regression (CRR) [62]. We further diversify the pool to make Sage's policy more stable.

In short, we maintain two neural networks, one for policy π_θ with parameter θ and one for the Q function Q_w parameterized by w. The learning algorithm consists of two iterative steps: policy evaluation and policy improvement. During the policy evaluation step, we want to learn Q_w to approximate the optimal Q function with D. We minimize the temporal difference error according to the following loss:

$$L(w) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}[d(Q_w(s_t, a_t), (r_t + Q_{w'}(s_{t+1}, \pi_{\theta'}(s_{t+1}))))] \tag{5}$$

where d measures the distance between the current $Q_w(s, a)$ and the Bellman update through the target networks (θ', w') as we use a distributional version of the Q update to stabilize learning [15].

In the policy improvement step, the agent produces a new policy π_θ based on Q_w and D. The policy improvement procedure encourages the agent to learn good actions from D and avoid taking unknown problematic actions. The objective can be written as optimizing a policy to match the state-action mapping in D, while regularized by the Q-function for better action:

$$\arg\max_{\pi} \mathbb{E}_{(s,a)\sim D}[f(Q_w, \pi, s, a) \log \pi(a|s)], \tag{6}$$

where f is a filter for action selection. We adopt $f = \exp(A_w(s, a)) = \exp(Q_w(s, a) - \frac{1}{m}\sum_j Q_w(s, a^j))$, with $a^j \sim \pi(\cdot|s)$. The intuition here is that the agent is motivated to learn a better sequence of actions from multiple behavior policies in multiple environments, as presented in D.

Moreover, to further improve the policy and stabilize learning, we use rich behavior policies to form D. In other words, we let D include different trajectories regardless of whether reward values associated with those trajectories are high or not. This makes the optimization of Eq. 5 more stable, and as data coverage increases, the estimation of the Q function can be improved. Since the learning algorithm iteratively learns Q and π, it will enhance the learning of the policies. We later show in section 7.5 the effectiveness of using rich policy pools to boost performance.

Putting all together, as a brief summary, first, we sample a mini-batch size $\mathcal{B}$ of samples $(s_t, a_t, r_t, s'_{t+1})$ from D. We update Q_w by minimizing Eq. 5. Then, we update π_θ by Eq. 6. We iteratively repeat the above steps until the training converges to an acceptable policy. Doing that, Sage can obtain a policy outperforming the pool of behavior policies in D.

5 TRAINING SAGE

We implement Sage's DNN and its Core Learning block using TensorFlow [2] and Acme [39]. We leverage the TensorFlow graph operation, which allows the model's inference to be made in real time. This addresses the challenges of using a deep architecture where model inference could become the bottleneck and ensure that the output action will feed into the underlying kernel timely.

The General-Purpose Clusters: In contrast with the state-of-the-art ML-based CC schemes (e.g., Orca) which require a large cluster of customized servers (either patched with new Kernel codes or require access to packet-level services) to effectively perform their training phases [9], the data-driven RL aspect of Sage enables its training sessions to be run over existing available general-purpose GPU clusters. This greatly facilitates the training phase and enables people with no luxury of accessing big custom clusters to still have impacts in this domain. We need to highlight that after training, Sage simply runs on top of a normal end-host, and discussions here were solely about the training phase.

Pool of Policies: Our pool of policies consists of two main sets: Set I and Set II. In Set I, we focus on the behavior of policies in

single-flow scenarios with respect to gaining high throughput and low delay, while in Set II, we collect their dynamics in the presence of TCP Cubic flows and their corresponding TCP-friendliness scores. In particular, to observe the behavior of different schemes in stable and changing conditions, Set I includes constant link capacity scenarios and scenarios in which link capacity changes. On the other hand, Set II includes scenarios where a competing TCP Cubic flow coexists with the CC scheme under the test. Overall, more than 1000 different environments are covered[5]. For more details and discussions about these sets, please see Appendix C. Finally, we use 13 available Internet CC schemes in the Linux Kernel: Westwood [20], Cubic [35], Vegas [17], YeAH [14], BBR2 [19], NewReno [34], Illinois [48], Veno [30], HighSpeed [28], CDG [36], HTCP [45], BIC [68], and Hybla [18] as the existing CC policies in our pool. Overall, our pool includes more than 60 million data points.

5.1 The Leagues, Scores, & Winning Rates

From now on, when we consider a certain group of CC schemes, we often refer to them as a league. To report the results and compare different schemes, we use a new terminology. In contrast to the usually used throughput-delay graphs that provide a qualitative comparison between the existing schemes, we employ a ranking terminology inspired by the straightforward rankings used in the games. In particular, we consider two main metrics and assign scores to different schemes based on these metrics. We rank the schemes in a certain scenario using these scores and identify so-called winner schemes. Any scheme gaining a score in the range of $[0.9, 1]\times$ the top score in a scenario, is considered a winner scheme. Later, we calculate the winning rate of individual schemes (the total number of wins over the total number of scenarios). Finally, we rank the schemes based on their winning rates. For the sake of space, we omit the discussions around the details of these definitions. For more details, please check Appendix D and [3].

Single-Flow Scenarios: To reflect the throughput and delay performance metrics of scheme c, we employ a form of Power defined by $S_p = \frac{r_c^\alpha}{d_c}$ where r_c and d_c are the average delivery rate and the average round-trip delay of c, respectively, and α is a coefficient determining the relative importance of throughput and delay[6]. A bigger value of S_p indicates a better performance for c.

Multi-Flow Scenarios: To reflect the TCP-friendliness metric, in a multi-flow scenario, we define $S_{fr} = |f_c - r_c|$ where f_c is the ideal average fair share of c, when competing with the flows with the default CC scheme, and r_c is the actual achieved average delivery rate of c. A smaller value of S_{fr} indicates a better TCP-friendliness.

5.2 Performance During Training

Using the defined terminologies, we report the overall performance of Sage during the training (done over a single GPU setting). In particular, after every roughly 24 hours of training, we record the gained model and collect its performance over all network environments used in Set I and Set II. Then, we compare Sage's winning rates against the league of CC schemes appeared in our pool of policies. Fig. 7 shows the results during one week of training. After 5 days, Sage enters the TCP-friendliness region[7], while surpassing the winning rates of all existing schemes in the single-flow scenarios. During the next days, Sage continues to increase its performance gap with existing heuristic policies. These promising results indicate that, indeed, machines can automatically learn from the existing heuristic designs, while not being bound by them.

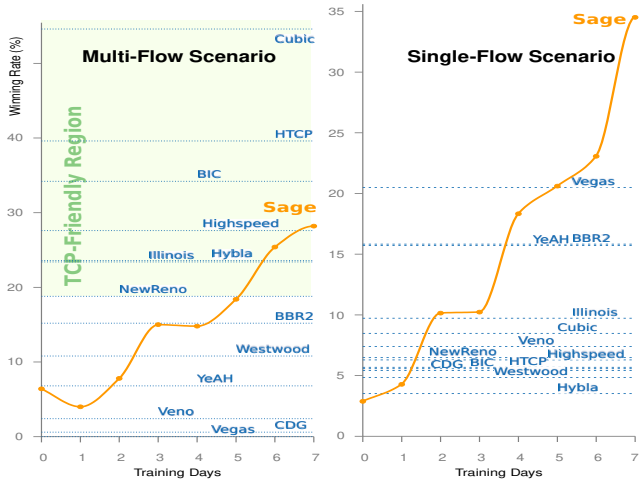


Figure 7: Performance of Sage during a 7-day training phase in single-flow (right) and multi-flow (left) scenarios

Notice: Note that, as mentioned in section 1, by design, Sage does not aim to be the *optimal* solution. Instead, it is designed with the aim of being *better* than existing ones. The fact that Sage does not achieve a 100% winning rate in Fig. 7 can highlight this.

6 GENERAL EVALUATION

Here, we compare Sage with different state-of-the-art schemes including the ones that did not appear in the pool[8]. In particular, in section 6.1, we assess Sage on Internet and in sections 6.2 and 6.3 we elaborate on the league of ML-based and delay-based schemes compared to Sage.

6.1 Consistent High-Performance

To examine the performance of Sage over complicated real networks and assess how general its gained model is, we perform three sets of evaluations: (1) Intra-continental, (2) Inter-continental, and (3) Highly variable networks. We highlight an important fact that all of these experiments were done after the model was learned and fixed, so the model was not tailored to any of these scenarios.

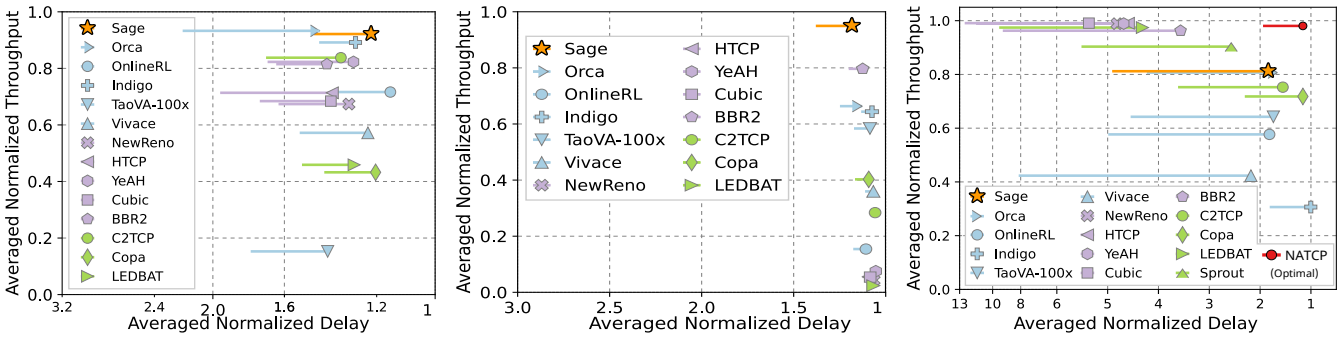
For the first two sets, we respectively employed different servers around the US continent (located in 16 different cities) and 13 different servers around the globe (outside of the US) representing different characteristics (e.g., the minimum RTTs spanning from

[5]We use an improved/debugged version of Mahimahi [52], which appeared in [3], as our base emulator.

[6]Unless otherwise mentioned, we set $\alpha = 2$. Please check Appendix D for a discussion about other values of α.

[7]We take TCP NewReno's winning rate in multi-flow scenarios as the base winning rate indicating a TCP-friendly region, due to its pure AIMD logic often used for modeling a simple general TCP flow.

[8]Because they do not provide APIs for collecting requried info. See section 8, for a discussion on extending the pool.



(a) Intra-Continental experiments (b) Inter-Continental experiments (c) Highly variable link experiments

Figure 8: Normalized avg. delay (icons), 95%tile delay (end of lines), and avg. throughput in different settings

7ms to 237ms) and sent traffic over the Internet among these servers. For the highly variable links, we emulated 23 cellular traces gathered in a prior work [9] and sent traffic using different schemes. For all these experiments, we repeat the tests five times and report the averaged normalized delay (over minimum gained delay) and averaged normalized throughput performance (over maximum gained throughput) of different schemes (for more details on the setup of the experiments please check Appendix G). Fig. 8 shows the overall results for different schemes averaged over all experiments. A sample set of the detailed version of these results is demonstrated in Appendix H. To have more readable plots, we omitted the bad-performing schemes.

Main Takeaways: Clearly, Fig. 8 illustrates the fact that each of the existing CC heuristic schemes fails in some scenarios, though may perform well in others. For instance, delay-based schemes (colored green), perform well in highly variable scenarios, while failing in Inter-continental ones. On the other hand, throughput-oriented schemes (colored violet) can do a good job in Intra-continental scenarios, while failing in highly variable ones. However, Sage works well in different complex unseen Internet scenarios and can outperform the heuristic schemes seen in the pool of policies. For instance, compared to BBR2, it achieves about 2× lower averaged delay in highly variable links, while in Inter-continental scenarios on average, it gains about 20% higher utilization. Also, Sage's learned policy can outperform the schemes that were not present in the pool. For instance, compared to the state-of-the-art learning-based scheme, Orca, Sage achieves 1.4× higher throughput, while having similar delay performance in Inter-continental scenarios. In sum, these promising results can suggest that Sage's deep model is not overfitted to specific scenarios during the training, and it can scale well to complex real-world unseen environments. That said and to be fair, we should mention that larger-scale in-field evaluations and measurements are always needed before making any concrete conclusions.

6.2 The League of ML-Based Schemes

To put the advantages of Sage's data-driven RL framework in a proper context, here, we compare sage with a league of ML-based CC schemes including 13 ML-based counterparts. In particular, we report the standings of different schemes in this league based on their winning rates in Set I and II. Fig. 9 shows the ranking of the league of ML-based designs (the dynamics of some of these schemes in some sample environments of Set II are depicted in Fig. 24).

1) Compred to BC: To elaborate on the issues of BC, we designed multiple BC counterparts of Sage that use the same input signals and are trained over the same duration and settings used for Sage by optimizing the log-likelihood loss on different pools of policies. In particular, we made four BC models: (1) BC-top (trained using top schemes of Set I and Set II), (2) BC-top3 (trained using policies of top three schemes of Set I and Set II), (3) BC (trained using all 13 schemes in Set I and Set II), and (4) BCv2 (trained using only the winner policies of each particular scenario in each Set). All different BC models perform poorly compared to Sage. This poor performance is based on two reasons. First, CC decisions/actions significantly impact future observations from a network. So, an approach such as BC that only tries to imitate the state-action mappings observed during the training phase faces severe issues later during the evaluation phase, where states are no more similar to the previously observed ones. Second, different CC schemes sometimes use contradictory strategies. So, an approach that tries only to clone these opposing strategies will fail in practice. This issue reveals itself when the pool of policies includes more contradictory policies, as in the BC model. These discussed issues can shed more light on the challenges and complexities involved in the design of Sage.

Remark: The strategy of imitating different heuristic schemes even in a large set of environments is far from sufficient.

2) Compared to Online RL: To have a fair comparison with the online RL approach, in addition to existing schemes (e.g., Aurora [42] which uses a simple online on-policy RL agent and considers only single flow reward and Genet [67] which attempts to improve Aurora by using curriculum learning), we designed an extra scheme called OnlineRL. OnlineRL is the online RL counterpart of Sage meaning it exploits the same input signals, employs the same reward functions, is trained for the same duration and over the same network environments of Set I and Set II, but in contrast with Sage, it utilizes the state-of-the-art online off-policy RL. As mentioned in section 1.1, online RL algorithms face issues when trained over a large set of environments/tasks. The results of OninerRL scheme clearly indicate that. Although OnlineRL can achieve an excellent

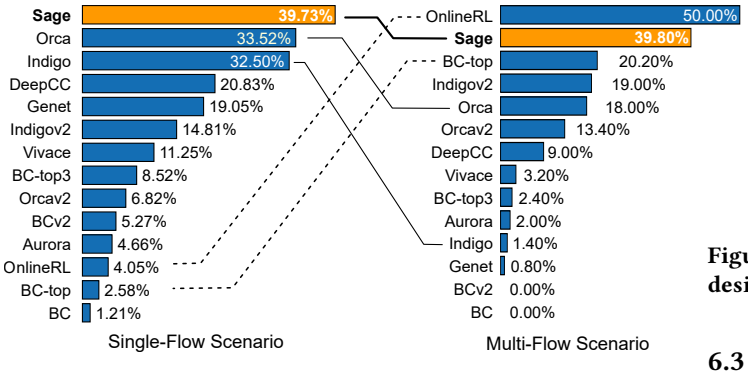


Figure 9: The ranking and winning rates of ML-based designs for the single-flow and multi-flow scenarios

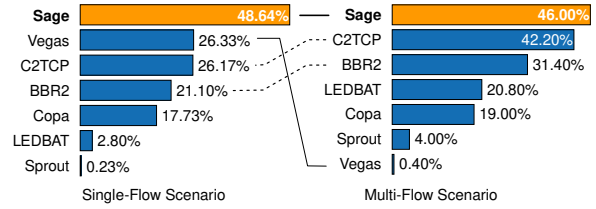


Figure 10: The ranking and winning rates of delay-based designs for the single-flow and multi-flow scenarios

winning rate in Set II, it significantly fails in Set I. Even with the luxury of interacting with all environments during the training, OnlineRL converges to an *unbalanced* model that works well only in a particular setting and very badly in others.

Remark: Online RL algorithms face serious issues during the training of large complex sets, and it is very difficult to converge to a policy that can, overall, perform very well.

3) Compared to Hybrid RL: Hybrid schemes, such as Orca [9] and DeepCC [10], aim to achieve high performance by combining a heuristic congestion control with online RL algorithms. In particular, our previous work, Orca, leveraging state-of-the-art online off-policy RL algorithms becomes the runner-up scheme in Set I. However, it gets about 2× lower winning rates compared to Sage in Set II. To make sure that this performance gap is not due to the different training settings or lack of a reward for multi-flow scenarios, we replaced Orca's reward with rewards used in Sage, we retrained Orca over the same environments of Set I and II for seven days. We call the gained model Orcav2. Performance of Orcav2 shows that optimizing for an extra reward over more large settings and for a longer duration can confuse Orca. Once again, this shows why a general impression about learning-based systems that assumes training for more scenarios and longer duration always boosts performance is wrong.

Remark: Even with the help of heuristic CC algorithms, online RL schemes face severe issues during the training of large complex sets in practice.

4) Compared to Imitation Learning: As an example of imitation learning, Indigo [69] attempts to approximate optimal CC oracles and then learn to imitate them in different settings. Although Indigo's single-flow performance is in third place, it falls to the last three in Set II. To be fair, Indigo's model was not trained over multi-flow scenarios. So, following the suggestions of Indigo's authors we added multi-flow scenarios and retrained it for 7 days (we call the new model Indigov2). As the results clearly show, a better performance of Indigov2 compared to Indigo in Set II comes with a great performance degradation (over 2×) in Set I.

Remark: Imitation learning comes with its own issues, such as the complexity of finding Oracles in different settings, unbalanced models, and confusion over complex environments.

6.3 The League of Delay-Based Schemes

Generally, Set I seems to be a perfect setting for revealing the advantages of delay-based CC schemes. Therefore, here we compare performance of Sage with the league of delay-based schemes including some recent ones: BBR2 [19], Copa [12], C2TCP [5, 7], LEDBAT [58], Vegas [17], and Sprout [66]. Fig. 10 shows the rankings of this league for both Set I and II (The dynamics of some of these schemes in some sample environments of Set II are depicted in Fig. 25). Sage shows great performance even when compared to these designs in Set I. In Set II with its completely different dynamics, Sage ranks first too. The bottom line is that, with Sage's data-driven method, the agent acquires the skill of incorporating two seemingly contradicting objectives (in Set I and II) in one model by effectively distinguishing network scenarios. This enables Sage to even surpass the performance of top delay-based schemes in scenarios where they are mainly designed for.

7 DEEP DIVE

7.1 Sage & Handling Distributional Shift

While Sage has trained under a certain data distribution, when evaluated, it can observe a different distribution. One of the main subtle reasons for that is the fact that actions performed by any CC scheme heavily impact the future observable states of the network. In other words, even when evaluated in the same environment where it has seen states corresponding to other schemes during training, Sage's observed states may differ from the dataset. To shed more light on this and demonstrate that Sage can handle these distributional shifts, here, we perform an experiment.

We choose an arbitrary environment from the pool (a step scenario where the BW changes from 24Mbps to 96Mbps), rollout Sage in this network for 30 seconds, and record the observed trajectories at every timestep t. To quantify the differences between Sage's observed states and existing ones in the pool, we define and utilize a metric called Distance. In particular, for the Sage's transition $u = (s_t, a_t, s_{t+1})$ at timestep t, we calculate the pairwise cosine distance[9] between u and existing vectors in the pool, and define the minimum of these pairwise distances as the Distnace of u. As the baseline, we run two other schemes (Vegas and BC) under the same setting and obtain the Distance of their trajectories. Fig. 11 shows the CDF of them.

Vegas is already one of the schemes in the pool and its new runs are expected to be similar to its previous ones. That is why

[9]The cosine distance of vectors u and v is defined as $1 - \frac{u \cdot v}{\|u\|\|v\|}$, where $u \cdot v$ denotes the dot product of u and v.

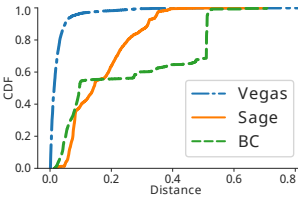


Figure 11: Distance's CDF

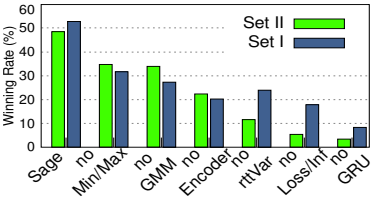


Figure 12: Ablation study

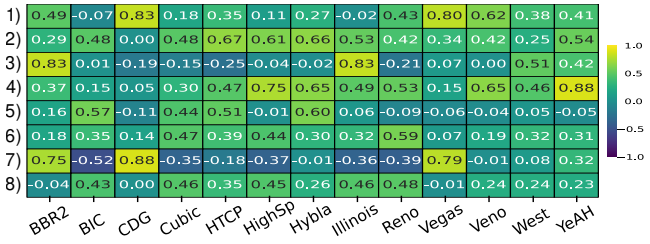


Figure 13: Sage's Similarity Indices to different schemes on eight random env. (each row represents an env.)

most of the time, the Distance values of Vegas are very low. On the other hand, BC and Sage observe different trajectories compared to the dataset (with 65%tile values of 0.2 and 0.45, respectively). This verifies that observed trajectories during the evaluation differ from the dataset. As discussed in section 6.2, this distributional shift greatly impacts BC and leads to its poor average throughput and delay (1.2Mbps, 43.2ms) performance. However, Sage effectively manages this shift and achieves the performance of (54.8 Mbps, 69.5ms) even higher than Vegas (36.1 Mbps, 76.1ms).

7.2 (Dis)Similarity of Sage to Other Schemes

Sage is trained solely by observing existing heuristics, so it might be natural to think that its model might have been biased toward certain schemes in the pool. To investigate that and illustrate that it is not the case, we choose eight different environments and send traffic using different CC schemes and gather trajectories of different schemes in the form of (s_t, a_t, s_{t+1}) vectors. To quantify the similarity between schemes, we use cosine similarity index defined as $\frac{u \cdot v}{\|u\|\|v\|}$ for two given vectors u and v. In particular, for any scheme A, we calculate the cosine similarity of Sage's trajectories to A's counterpart trajectories in that environment and calculate the average of these values over all trajectories of Sage. We call the gained value the Similarity Index of Sage to A (with 1 showing perfect similarity). Results are reported in Fig. 13 (each row presents the Similarity Indices in one environment).

Considering these different environments, Fig. 13 highlights different points. The main one is that Sage's model is not biased toward one or even a certain few schemes in the pool, because the schemes with the highest Similarity Indices change widely in different environments. In other words, each of these 13 schemes has impacted the final model to different degrees. Moreover, the value of the highest Similarity Index itself changes in different settings. That can be due to the fact that Sage is not bound by these schemes in these environments and has learned a policy that is not simply a cloned version of them in different settings[10].

7.3 Ablation Study

Here, we perform ablation studies to assess the importance of various components of Sage including the role of main blocks in its NN and the impact of different input signals. To that end, we retrain 6 more models under the same 7-day training regime and report their performance in Fig. 12.

Value of Different Input Signals: To elaborate on the role of various input signals, we removed three different sets of signals leading to these models: (1) no Min/Max model [removing all min/max statistics from input leading to a vector of 33 elements as input], (2) no rrtVar model [removing rates and variances of RTT values (rows 23-40 in Table 1)], (3) no Loss/Inf model [removing loss and inflight information (rows 41-58) in Table 1)]. Fig. 12 clearly shows removing different parts of the input signal degrades the performance. In particular, loss and RTT variance-related information are key in multi-flow scenarios. This can be enlightening info even for heuristic CC designers!

Sage's NN Architecture: To highlight the importance of Sage's NN architecture, we made three other variations leading to these models: (1) no GRU model [by removing the GRU block], (2) no Encoder model [by removing the encoder block right after the GRU], (3) no GMM model [by replacing the GMM unit of the last layer with multivariate normal distribution]. As it is clear, each of these components helps increase the effectiveness of training from the pool of existing policies. In particular, GRU block plays a crucial role for Sage. This is based on the fact that GRU works as a form of memory to capture sequence-level information from data.

7.4 Impact of Input Representation

Now, we investigate the importance of different time granularities used by the GR block to generate the input vector. To that end, we vary the granularity of calculating input statistics and reconstruct three pools: Small [observation window of 10], Medium [observation window of 200], and Large [observation window of 1000]. Using these new pools and the same 7-day training regime, We gain 3 models: Sage-s, Sage-m, and Sage-l, respectively, and report their performance compared with Sage in Fig 14. To provide more insight, we employ a visualization technique (t-SNE algorithm [61]) to visualize the output of the models' last hidden layer when evaluating them over seven environments sampled from Set II (Fig. 16). Results highlight different points. First, when a long-term goal (TCP-friendliness) is considered, inputs that reflect the history of the network more lead to higher winning rates. This is why the winning rate in Set II for Sage-l is much better than Sage-m and Sage-s. Also, visualization results of Fig. 16 confirm that Sage-s and Sage-m have a hard time distinguishing multi-flow environments, while Sage-l's hidden representations can effectively separate them. Second, results of Set I suggest that satisfying myopic objectives depends more on a balanced combination of the current state and the network history. Third, the distribution of the winning rate in Set I indicates that there are still scenarios in which each of these granularities can benefit the system. In particular, even Sage-s can

[10]We leave a more detailed analysis of Sage's similarity/dissimilarity to other schemes for future work.

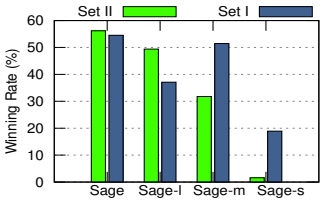


Figure 14: Impact of different input representations

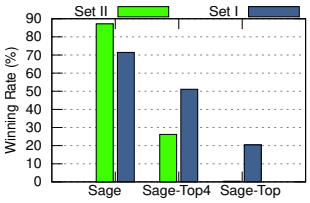


Figure 15: Impact of the diversity of the pool

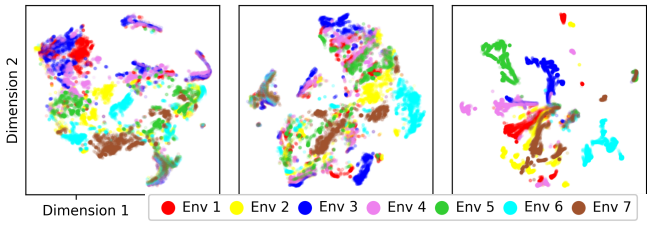


Figure 16: The t-SNE [61] representation of the last hidden layer of Sage-s (left), Sage-m (middle), & Sage-l (right) for seven diff. env. chosen randomly from Set II

still be a winner in about 20% of cases in Set I. Fourth, although the setup was very much in favor of the other three rivals, Sage still performs very well (consider that the total number of inputs for each of these 3 rivals (= 33 signals) is much smaller than Sage (= 69 signals), while the duration of training was equal for all of them).

The bottom line is that for learning a better policy, it is critical to provide the agent with proper general input representations for effectively extracting valuable knowledge from a given high-dimensional input.

7.5 The More the Merrier

In this section, we highlight two points: (1) learning from only the best policy is not sufficient and (2) as a data-driven approach, Sage can benefit from a more diverse pool of policies. To that end, we retrain Sage with a couple of alternative pools of policies. For the first pool, we only include the top-ranked schemes of Set I and II (Vegas and Cubic) and name the gained model Sage-Top. For the second pool, we include the four top-ranked schemes of Set I and II ({Vegas, BBR2, YeAH, Illinoise} and {Cubic, HTCP, BIC, Highspeed}) and name the gained model Sage-Top4. Fig. 15 shows the results for these models compared to Sage. Even with the same number of data points, the model trained using a smaller number of policy variations performs worse than the one that utilized a more diverse pool during the training. This clearly shows the two mentioned remarks. The key here is that Sage benefits from observing more diverse policies, even schemes that do not perform well overall.

7.6 Sage's Behavior in Three Sample Scenarios

To provide more insight, we demonstrate how Sage reacts in three sample environments: 1) when the link capacity suddenly doubles (24Mbps to 48Mbps), 2) when the link capacity suddenly halves (48Mbps to 24Mbps), and 3) in the presence of a Cubic flow (24Mbps

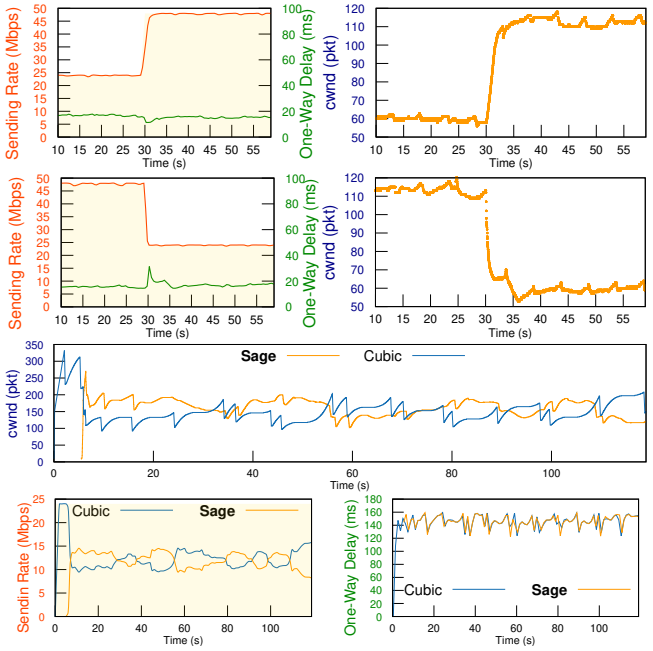


Figure 17: Sage's sending rate, one-way packet delay, and cwnd in 3 different sample scenarios: (row #1) a sudden increase in link capacity, (row #2) a sudden decrease in link capacity, and (rows #3 & #4) competing with a Cubic flow.

link). For these experiments, we set the minimum delay of 20ms and bottleneck buffer size of 450KBytes (=300pkts). Results of sending rates, cwnd, and one-way packet delays of Sage across time are presented in Fig. 17. The results highlight some interesting behaviors of Sage. For example, the changes in Sage's cwnd in the 1st and 2nd scenarios can suggest that Sage has learned a probing policy for discovering available bandwidth without causing excessive delay increases. In these two scenarios, after settling down on a proper cwnd value, Sage keeps performing a somewhat 5-second periodic probing. In fact, this probing helps it discover sudden available link capacity in 1st scenario. Interestingly, when Sage detects extra available capacity at t=30s, instead of conducting another 5-second probing, it continuously increases the sending rate until the link is fully utilized. Subsequently, it appears to revert to the probing phase. On the other hand, in the 3rd scenario realizing that it is not the only flow in the network, Sage behaves differently compared to the other two scenarios and attempts to compete fairly with a throughput-oriented scheme.

Notice: Note that these descriptions should not be interpreted as the exact representation of Sage's learned policy. The learned policy appears to be more complicated than that and varies in different scenarios. That said, we leave a detailed analysis of Sage's behavior for future work.

7.7 More on Dynamics of Sage

Fairness: How Sage behaves in the presence of other Sage flows? To investigate this question, we send traffic from our server to a randomly selected remote server on GENI. Then, every 25 seconds,

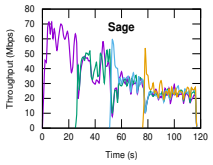


Figure 18: Sage's fairness

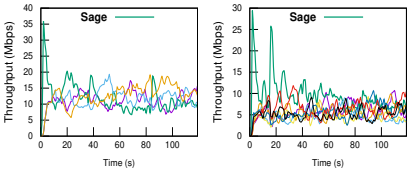


Figure 19: Sage's TCP-fr. (competing with 3 (left) and 7 (right) Cubic Flows)

we add another flow destined for the same server and measure the throughput of each flow over a 2-minute period. We repeat all the runs three times. Fig. 18 shows the fairness dynamics of Sage. Also, to put the results in a proper context, the fairness aspect of other schemes in the same setting is reported in Fig. 27. Sage has not trained over any scenario targeting directly the fairness objective. However, it still gains very good fairness property. This comes from the fact that fairness is a relatively simpler objective than TCP-friendliness, because the rival flows use the same CC strategy. A policy that has learned to compete fairly with aggressive loss-based schemes (while minimizing its delay when it is alone) will most likely perform a good job while competing with its own family of flows.

TCP-Friendliness: In the pool of our policies, we had two-flow scenarios. So, to investigate the performance of Sage's learned policy in other scenarios, we increase the number of competing Cubic flows. In particular, we emulate a 48Mbps, 40ms mRTT, and BDP buffer size bottleneck, share it between a Sage flow and three (and in another case, seven) more Cubic flows, and measure the throughput of each individual flow over a 2-minute period. Fig. 19 shows the results. Sage's performance can be appreciated more when the results of other schemes in the same setting are also considered. To that end, we report the results of some other schemes in Fig. 28 in Appendix J. Although Sage has not seen the behaviors of any existing policies in the presence of a large number of competing Cubic flows, the learned policy performs well and adapts to these new settings.

For more deep dive experiments including very low impact of different AQM schemes on Sage's performance and its performance frontier behavior compared to the pool, please check Appendix E.

8 LIMITATIONS & FINAL NOTE

Extending the Dataset: Sage provides more flexibility for training a CC policy by decoupling data collection and policy learning. In principle, Sage can train on ***any*** data gathered from not only heuristic schemes but also ML-based ones. The only requirement is that the Policy Collector should have access to general APIs to collect statistics of the underlying CC scheme. However, unlike kernel-based implementations, user-space CC implementations (almost most of the academic ML-based CC schemes) do not provide such APIs. That said, we suggest that CC designers add a simple kernel-like socket option API in their codes. Such a simple improvement will enable Sage (and similar future schemes) to extend their pool. That way, Sage can also learn from *your* ML-based or heuristic design and even from CC schemes executed on particular network scenarios that are hard to simulate/emulate or in which safety is a major concern.

Simplifying the Design and Lowering the Overhead: Our current model uses a deep architecture to learn a better-performing policy. This deep model, which runs in user space and real-time, can lead to CPU overhead. Although the added overhead is much smaller than most of the existing ML-based and even heuristic CC schemes, further optimizations are required before it can be utilized in practice[11]. That said, there are orthogonal active lines of work both in machine learning and network communities focusing on reducing the overhead of ML-based systems, including pruning during training to reduce redundant units of neural networks [29, 73], using quantization [22], knowledge distillation [38], and using a light deployment of neural networks for Kernel datapaths [72]. The bottom line is that these orthogonal efforts can help reduce the overhead of deploying NN-based systems, including Sage.

Generalization and Performance Guarantee: Although we showed that Sage's learned policy can keep its competitive performance in various complex scenarios tested including on the Internet, how general Sage's policy is, remains a question. This question and, in general, how a DRL/data-driven DRL model generalizes is still a very active research topic [32]. That said, some recent works use techniques such as curriculum learning to better train RL-based designs in practice (e.g., [67]). On the other hand, some other works focus on theoretical aspects and study provable generalization in RL, though they require strong assumptions such as a discrete action space, a low-dimensional input, or a linear model [25, 49]. The results of works on this track can also shed more light on Sage's generalization property.

Analysing Learning-based CCs: Why exactly a DNN makes a specific decision as it does? This is also another fundamental question that has not yet been fully answered. That said, some recent works propose tools for analyzing DNN (e.g.,[27]) or building more interpretable models (e.g.,[51]). Works on these orthogonal tracks can help to analyze ML-based systems such as Sage easier, though applying these or similar methods to complex deep architectures or high-dimensional inputs is not straightforward.

Final Note: A pure data-driven approach enables Sage to "Stand on the shoulders of giants" and demonstrate that machines can automatically exploit the vast pool of existing heuristic Internet CC schemes to discover better-performing ones. Sage by no means is the final solution to CC design, but we hope that it paves the way for a more sustainable way of designing Internet CC schemes and also show that learning-based systems are not necessarily archenemies of heuristic ones.

ACKNOWLEDGMENT

We would like to thank our shepherd Justine Sherry and anonymous reviewers whose comments helped us improve the paper. Chen-Yu thanks NYU IT for providing HPC resources for training. Soheil would like to express gratitude to Yashar Ganjali and Compute Canada for their generous provision of access to general-purpose GPU clusters, which were instrumental in training Sage.

[11]For instance, when sending traffic over a 200Mbps link, compared to Aurora (an online RL-based design) and Copa (a simple heuristic), Sage achieves more than 2× and 1.5× lower overhead, respectively.

REFERENCES

[1] 2023. Sage's Public Repository. https://github.com/soheil-ab/sage/. (2023).

[2] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, et al. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[3] Soheil Abbasloo. 2023. Internet Congestion Control Benchmarking. https://doi.org/10.48550/arXiv.2307.10054. *arXiv preprint arXiv:2307.10054* (2023).

[4] Soheil Abbasloo and H Jonathan Chao. 2019. Bounding Queue Delay in Cellular Networks to Support Ultra-Low Latency Applications. *arXiv preprint arXiv:1908.00953* (2019).

[5] Soheil Abbasloo, Tong Li, Yang Xu, and H. Jonathan Chao. 2018. Cellular Controlled Delay TCP (C2TCP). In *IFIP Networking Conference (IFIP Networking), 2018*.

[6] Soheil Abbasloo, Yang Xu, and H. Jonathan Chao. 2018. HyLine: a Simple and Practical Flow Scheduling for Commodity Datacenters. In *IFIP Networking Conference (IFIP Networking), 2018*.

[7] 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.

[8] Soheil Abbasloo, Yang Xu, H Jonathon Chao, Hang Shi, Ulas C Kozat, and Yinghua Ye. 2019. Toward Optimal Performance with Network Assisted TCP at Mobile Edge. *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)* (2019).

[9] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2020. Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet. In *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM '20)*. 632–647.

[10] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2021. Wanna make your TCP scheme great for cellular networks? Let machines do it for you! *IEEE Journal on Selected Areas in Communications* 39, 1 (2021), 265–279.

[11] 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.

[12] 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.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450* (2016).

[14] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. 2007. YeAH-TCP: yet another highspeed TCP. In *Proc. PFLDnet*, Vol. 7. 37–42.

[15] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*. PMLR, 449–458.

[16] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61 (2014), 5–23. Special issue on Future Internet Testbeds – Part I.

[17] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM '94)*. 24–35.

[18] Carlo Caini and Rosario Firrincieli. 2004. TCP Hybla: a TCP enhancement for heterogeneous networks. *International journal of satellite communications and networking* 22, 5 (2004), 547–566.

[19] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2019. TCP BBR v2 Alpha/Preview Release. https://github.com/google/bbr/blob/v2alpha/README.md. (2019).

[20] Claudio Casetti, Mario Gerla, Saverio Mascolo, Medy Y Sanadidi, and Ren Wang. 2002. TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks* 8, 5 (2002), 467–479.

[21] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[22] Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. 2021. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *Proceedings of Machine Learning and Systems* (2021), 873–884.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[24] 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.

[25] Simon Du, Sham Kakade, Jason Lee, Shachar Lovett, Gaurav Mahajan, Wen Sun, and Ruosong Wang. 2021. Bilinear classes: A structural framework for provable generalization in rl. In *International Conference on Machine Learning*. PMLR, 2826–2836.

[26] Richard O Duda and Peter E Hart. 1973. *Pattern classification and scene analysis*. Vol. 3. Wiley New York.

[27] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. 2021. Verifying Learning-Augmented Systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. 305–318.

[28] Sally Floyd. 2003. RFC3649: HighSpeed TCP for Large Congestion Windows. (2003).

[29] Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*.

[30] Cheng Peng Fu and Soung C Liew. 2003. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on selected areas in communications* 21, 2 (2003), 216–228.

[31] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*. PMLR, 2052–2062.

[32] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. 2021. Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability. In *Advances in Neural Information Processing Systems*.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[34] Andrei Gurtov, Tom Henderson, and Sally Floyd. 2004. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782. (2004).

[35] 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.

[36] David A Hayes and Grenville Armitage. 2011. Revisiting TCP congestion control using delay gradients. In *International Conference on Research in Networking*. Springer, 328–341.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.

[38] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[39] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, et al. 2020. Acme: A Research Framework for Distributed Reinforcement Learning. *arXiv preprint arXiv:2006.00979* (2020).

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

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

[42] 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*. PMLR, 3050–3059.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[44] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*.

[45] Douglas Leith and Robert Shorten. 2004. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet*, Vol. 2004.

[46] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).

[47] Fenglin Liu, Xuancheng Ren, Zhiyuan Zhang, Xu Sun, and Yuexian Zou. 2020. Rethinking skip connection with layer normalization. In *Proceedings of the 28th international conference on computational linguistics*. 3586–3598.

[48] 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.

[49] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. 2020. Provably Good Batch Off-Policy Reinforcement Learning Without Great Exploration. In *Advances in Neural Information Processing Systems*.

[50] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. 2022. Multi-Objective Congestion Control. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*. 218–235.

[51] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting Deep Learning-Based Networking Systems. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. 154–171.

[52] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 417–429.

[53] Kathleen Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (2012), 42–50.
[54] 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 *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 148–155.
[55] 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 (CoNEXT '18)*. 307–319.
[56] Dean A. Pomerleau. 1988. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Advances in Neural Information Processing Systems*.
[57] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. 2022. A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems. *arXiv preprint arXiv:2203.01387* (2022).
[58] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. 2010. LEDBAT: The New BitTorrent Congestion Control Protocol.. In *ICCCN*. 1–6.
[59] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An Experimental Study of the Learnability of Congestion Control. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. 479–490.
[60] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *IEEE INFOCOM 2006*. 1–12.
[61] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).
[62] Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. 2020. Critic Regularized Regression. In *Advances in Neural Information Processing Systems*.
[63] Wikipedia. 2023. Chess Title. https://en.wikipedia.org/wiki/Chess_title#\protect\protect\leavevmode@ifvmode\kern+.2222em\relax~\protect\protect\leavevmode@ifvmode\kern+.2222em\relaxtext=federations%20as%20well.-,Master,%27norms%27%20during%20tournament%20play. (2023).
[64] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 50–61.
[65] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the 2013 Conference of the ACM SIGCOMM (SIGCOMM '13)*. ACM, 123–134. https://doi.org/10.1145/2486001.2486020
[66] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks.. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 459–471.
[67] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. 2022. Genet: Automatic Curriculum Generation for Learning Adaptation in Networking. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. 397–413.
[68] Lisong Xu, Khaled Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE INFOCOM 2004*, Vol. 4. 2514–2524.
[69] 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.
[70] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. 2021. ACC: Automatic ECN Tuning for High-Speed Datacenter Networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. 384–397.
[71] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *ACM SIGCOMM CCR*, Vol. 45. ACM, 509–522.
[72] Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen. 2022. LiteFlow: Towards High-Performance Adaptive Neural Networks for Kernel Datapath. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. 414–427.
[73] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
[74] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. 2021. Network Planning with Deep Reinforcement Learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. 258–271.

Appendices

Appendices are supporting material that has not been peer-reviewed.

A A PLETHORA OF SCHEMES

General CC: End-to-end CC algorithms typically compute a congestion window to determine how many packets should be sent to the network at the sender side. TCP-Reno [40] and TCP-NewReno [34] use the famous loss-oriented AIMD mechanism to adjust congestion windows. HighSpeed TCP[28] targets large BDP networks. TCP Hybla [18] focuses on networks with large intrinsic RTTs, such as satellite networks. Delay-based CC algorithms, such as TCP Vegas [17] and LEDBAT[58] use delay as a key congestion signal. Compound TCP [60] and TCP-Illinois [48] combine loss and delay signals to perform a better congestion management. Some other CC algorithms attempt to use certain models for the network and derive the target sending rates based on that. As an example of these white-box approaches, BBR2 [19] models the network with a simple single bottleneck link and estimates the BDP by continuously measuring the minimum delay and maximum throughput of the network. Copa [12] targets low-delay communication while attempting to be competitive with loss-based CC schemes in practice. All these schemes, in the end, turn out to perform well only in specific scenarios and fail in others.

Learning-based CC: Learning-based CC schemes let the machine generate a proper response to the congestion in the network. As an early attempt at computer-generated CC, Remy [65] uses a policy optimization that iteratively searches for the optimal state-action mapping table. A follow-up work [59] uses Remy as a tool to discuss the learnability aspect of CC schemes in more settings. RemyCC schemes, as already shown in different works, degrade dramatically when the evaluation scenarios diverge from previously trained networks, because Remy relies on very accurate assumptions about the underlying network and competing traffic. Indigo [69] uses imitation learning from oracles, which requires explicit knowledge and assumption of the underlying network to generate a CC response. It is unclear how to define the oracle considering more complicated network scenarios. Recently, some works have leveraged modern DRL for learning CC policies. Aurora [42] learns policy using vanilla RL, which requires extensive online access to the environments. Orca [9], first to highlight the practical issues of clean-slate learning-based techniques such as overhead, convergence issues, and low performance over unseen network conditions, combines heuristic CC designs with DRL and proposes a hybrid pragmatic system to address those issues. DeepCC [10] employs the same hybrid design philosophy, but instead of targeting to be a new CC scheme, it introduces a DRL-driven CC plug-in that automatically boosts the performance of a given CC scheme in highly variable cellular scenarios without requiring to change the underlying CC scheme. MOCC [50] uses RL to adapt to different application objectives. Among other works, some (e.g., Vivace [24]) utilize online-learning for CC. We have compared Sage in detail with most of these learning-based schemes in section 6.

Specialized CC: Some CC algorithms focus on specific networks with their unique characteristics. Sprout [66], Verus [71], NATCP [8], C2TCP [7], and ExLL [55] focus on cellular networks.

Table 1: Sage's input vector and a short description of its elements

Index	Input Statistic	Description	Index	Input Statistic	Description
1	srtt	smoothed RTT (sRTT) cal. by Kernel	36	rtt_var_m.min	min(rtt variance) over Medium time wnd
2	rttvar	variance of sRTT	37	rtt_var_m.max	max(rtt variance) over Medium time wnd
3	thr	current delivery rate	38	rtt_var_l.avg	avg(rtt variance) over Large time wnd
4	ca_state	socket ca state, e.g., OPEN, LOSS, etc.	39	rtt_var_l.min	min(rtt variance) over Large time wnd
5	rtt_s.avg	avg(srtt) over Small time wnd	40	rtt_var_l.max	max(rtt variance) over Large time wnd
6	rtt_s.min	min(srtt) over Small time wnd	41	inflight_s.avg	avg(unack bytes) over Small time wnd
7	rtt_s.max	max(srtt) over Small time wnd	42	inflight_s.min	min(unack bytes) over Small time wnd
8	rtt_m.avg	avg(srtt) over Medium time wnd	43	inflight_s.max	max(unack bytes) over Small time wnd
9	rtt_m.min	min(srtt) over Medium time wnd	44	inflight_m.avg	avg(unack bytes) over Medium time wnd
10	rtt_m.max	max(srtt) over Medium time wnd	45	inflight_m.min	min(unack bytes) over Medium time wnd
11	rtt_l.avg	avg(srtt) over Large time wnd	46	inflight_m.max	max(unack bytes) over Medium time wnd
12	rtt_l.min	min(srtt) over Large time wnd	47	inflight_l.avg	avg(unack bytes) over Large time wnd
13	rtt_l.max	max(srtt) over Large time wnd	48	inflight_l.min	min(unack bytes) over Large time wnd
14	thr_s.avg	avg(thr) over Small time wnd	49	inflight_l.max	max(unack bytes) over Large time wnd
15	thr_s.min	min(thr) over Small time wnd	50	lost_s.avg	avg(losts bytes) over Small time wnd
16	thr_s.max	max(thr) over Small time wnd	51	lost_s.min	min(losts bytes) over Small time wnd
17	thr_m.avg	avg(thr) over Medium time wnd	52	lost_s.max	max(losts bytes) over Small time wnd
18	thr_m.min	min(thr) over Medium time wnd	53	lost_m.avg	avg(losts bytes) over Medium time wnd
19	thr_m.max	max(thr) over Medium time wnd	54	lost_m.min	min(losts bytes) over Medium time wnd
20	thr_l.avg	avg(thr) over Large time wnd	55	lost_m.max	max(losts bytes) over Medium time wnd
21	thr_l.min	min(thr) over Large time wnd	56	lost_l.avg	avg(losts bytes) over Large time wnd
22	thr_l.max	max(thr) over Large time wnd	57	lost_l.min	min(losts bytes) over Large time wnd
23	rtt_rate_s.avg	avg(rtt_rate) over Small time wnd	58	lost_l.max	max(losts bytes) over Large time wnd
24	rtt_rate_s.min	min(rtt_rate) over Small time wnd	59	time_delta	time elapse b.w. two timesteps norm. to mRTT
25	rtt_rate_s.max	max(rtt_rate) over Small time wnd	60	rtt_rate	ratio of two recent rtts
26	rtt_rate_m.avg	avg(rtt_rate) over Medium time wnd	61	loss_db	current rate of newyly lost bytes
27	rtt_rate_m.min	min(rtt_rate) over Medium time wnd	62	acked_rate	normalized rate of Ack reception
28	rtt_rate_m.max	max(rtt_rate) over Medium time wnd	63	dr_ratio	ratio of two recent delivery rates (thr)
29	rtt_rate_l.avg	avg(rtt_rate) over Large time wnd	64	bdp_cwnd	ratio of current BDP over current cwnd
30	rtt_rate_l.min	min(rtt_rate) over Large time wnd	65	dr	delivery rate calculated by Kernel
31	rtt_rate_l.max	max(rtt_rate) over Large time wnd	66	cwnd_unacked_rate	ratio of unack packet over sent ones
32	rtt_var_s.avg	avg(rttvar) over Small time wnd	67	dr_max	maximum delivery rate
33	rtt_var_s.min	min(rttvar) over Small time wnd	68	dr_max_ratio	ratio of two adjacent dr_max
34	rtt_var_s.max	max(rttvar) over Small time wnd	69	pre_act	previous action
35	rtt_var_m.avg	avg(rttvar) over Medium time wnd			

These solutions exploit the knowledge of the specific characteristics of target networks during their design phase to better optimize their solutions to them. DCTCP [11], D3 [64], and HyLine [6] are other examples from this category that rely on data center networks' characteristics, such as their single-authority nature.

B SAGE'S INPUT SIGNAL

Table 1 shows the 69 input signals used by Sage and a brief description of them.

C MORE ON SET I AND SET II

C.1 Set I

This set consists of single-flow scenarios where schemes are monitored with respect to S_p score that reflects their throughput and delay performance. The Set I includes two main classes of scenarios: (1) the flat scenarios and (2) the step scenarios.

The Flat Scenarios: This set of scenarios represents general wired scenarios on the Internet. As its name suggests, it includes wired links with constant/flat bandwidths throughout the experiments. The ranges of BW, minRTT, and queue size (qs) are $[12, 192]$Mbps, $[10, 160]$ms, $[\frac{1}{2}, 16]\times$BDP, respectively.

The Step Scenarios: The flat scenarios alone cannot grasp the performance of CC schemes over a more dynamic network. So to answer questions such as how a CC policy behaves when BW reduces or increases suddenly, we bring up the step scenarios. In these scenarios, we start with a given network BW (BW_1) and after a specific period of time, we change the underlying BW of the network to $m \times BW_1$. The m value is chosen from $(0.25, 0.5, 2, 4)$ list. We observed that for large values of BW, Mahimahi's overhead increases to a point that it tangibly impacts the results. To prevent these unwanted impacts, when changing BW, we always choose to be under 200Mbps. This means that if BW_1 is 96Mbps, we choose $m < 4$. The range of other parameters is similar to flat scenarios.

C.2 Set II

This set provides scenarios for observing the TCP-friendliness aspect of Internet CC policies. To that end, we let TCP Cubic, which is the default CC scheme in most platforms (including Linux, Windows, and macOS), compete with the CC scheme under the test for accessing a shared bottleneck link and we capture the behavior of schemes with respect to S_{fr} score. Similar to the Set I, the three main network parameters are changed to make different scenarios. The ranges of minRTT and BW values are similar to Set I. In addition, we at least let the bottleneck link have 1×BDP buffer size to be able to effectively absorb more than one flow during the tests. In particular, we choose qs from $[1, 16]\times$BDP range.

In a general Internet scenario, with a good probability, we can assume that a new incoming flow will observe flows controlled by the default CC scheme on the bottleneck link. This comes from the definition of a default CC scheme and its property of being used

by the majority of flows. Therefore, we let TCP Cubic come to the network earlier than the CC scheme under the test. When buffer size increases, generally, it takes more time for flows to reach the steady state (if any). In our experiments, we observed that reaching a fair-share point may take more than a minute (even when both flows are Cubic flows). So, in Set II, we let flows send their packets for 120s to make sure that the results can present meaningful TCP-friendliness scores.

D THE NOTION OF SCORES & WINNING RATES

A more classic way of looking at who should be called the winner in a certain scenario may lead us toward recognizing the CC scheme with the best score gained throughout a scenario as the winner in that scenario. However, there are two issues with this way of identifying a winner.

First, since the scores defined in section 5.1 (S_p and S_{fr}) are Real numbers, their absolute values can differ slightly for two CC schemes. So, if we simply perform a mathematical comparison between scores, these slight differences can impact the choice of the winner in a scenario. That said, instead of picking the CC scheme with the best score as the winner, we pick all CC schemes with scores less than 10% worse than the best score as the winners of a scenario. In other words, any scheme with at most 10% lower performance than the best-performing scheme is included in the winner list of that scenario.

Second, simply assigning a number to the performance of a scheme over an entire scenario and then comparing these numbers together to decide the winners may smooth out the important differences among the CC schemes. For instance, how fast CC schemes can react to a sudden change in the network may not be visible in an overall score of the scheme over a longer period. To address this issue, we calculate the score of a scheme in separate intervals throughout the experiments and instead of one score, assign four scores corresponding to the performance of the scheme in four different intervals throughout the test. Now, comparing the scores of a certain interval for all schemes can get a better sense of the performance of different schemes.

Putting all together, we compare the scores of all CC algorithms over a certain interval of a certain scenario and pick the best-performing schemes (considering the 10% winning margins) as the winners. Then, we sweep over all intervals and scenarios.

D.1 Rankings with other Metrics

The choice of $\alpha = 2$ on the S_p score indicates that gaining $\sim 1.4\times$ higher throughput is equivalent to gaining $2\times$ lower delay. We think that this shows a more pragmatic metric than simply considering $1.4\times$ better throughput equal to gaining $1.4\times$ lower delay for the end users in the Internet. Moreover, to show that changing the values of α does not change the rankings provided that much, here, we provide the rankings of the delay-based and ML-based schemes for $\alpha = 3$. Tables 2 and 3 show the new rankings.

D.2 A Tighter Winning Margin

As discussed before, we mainly use a winning margin of 10% to identify the winners in different schemes. That means any scheme with a score in the $[0.9, 1]$ range of the best score in a certain environment is considered a winner scheme. Here, we examine a tighter margin to identify the winners. In particular, we use a 5% margin. As the results reported in Fig. 20 and 21 show, the rankings will remain largely intact when compared with a 10% winning margin.

Table 2: Winning Rates of schemes in the delay-based league in scenarios of Set I when winning metric is $\frac{r^3}{d}$

Rank	Scheme	Set I
#1	Sage	48.48%
#2	Vegas	25.27%
#3	C2TCP	24.85%
#4	BBR2	20.15%
#5	Copa	11.40%
#6	LEDBAT	02.54%
#7	Sprout	00.08%

Table 3: Winning Rates of schemes in the learning-based league in scenarios of Set I when metric is $\frac{r^3}{d}$

Rank	Scheme	Set I
#1	Sage	38.03%
#2	Orca	31.70%
#3	Indigo	27.58%
#4	DeepCC	20.08%
#5	Indigov2	15.27%
#6	Genet	14.24%
#7	Vivace	8.60 %
#8	BC-top3	8.22 %
#9	BCv2	5.42 %
#10	Orcav2	5.15 %
#11	Aurora	4.24 %
#12	OnlineRL	4.05 %
#13	BC-top	2.39 %
#14	BC	1.02 %

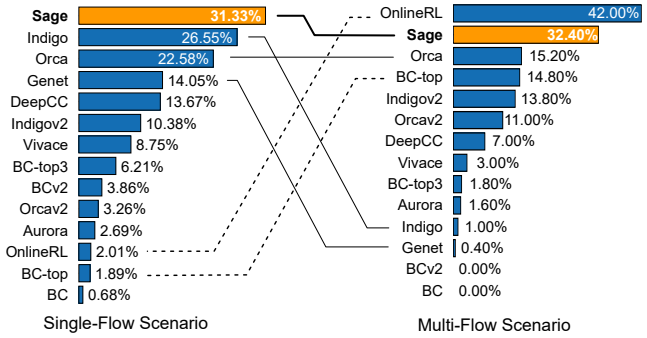


Figure 20: The ranking of the league of ML-based designs based on the schemes' winning rates for the single-flow (left) and multi-flow (right) scenarios when winning rate margin is 5% (instead of the default 10%)

E MORE ON DYNAMICS OF SAGE

E.1 Sage as the Performance Frontier

The learning goal of Sage is to harness existing heuristic schemes to automatically discover a better policy. To further highlight the

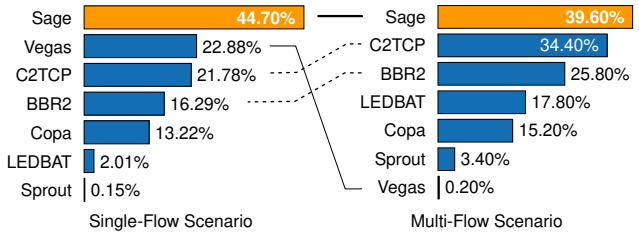


Figure 21: The ranking of the league of delay-based designs based on the schemes' Winning Rates for the single-flow (left) and multi-flow (right) scenarios when winning rate margin is 5% (instead of the default 10%)

strength of Sage, we compare Sage to thirteen heuristics that Sage has learned from. We evaluate Sage in two network environments where the link capacities are constant, one with a shallow buffer and the other with a deep buffer. Then, we report the data points depicting the performance of heuristic schemes in terms of throughput and delay in Fig. 22. As Fig. 22 illustrates, while exposed to a wide distribution of data, Sage can learn to operate in high-utilization and low-latency region and be the performance frontier. This highlights Sage's ability to automatically recover a better policy from the pool of existing heuristics with heterogeneous behaviors.

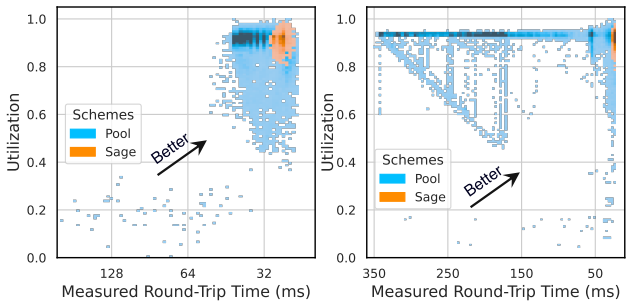


Figure 22: Sage automatically learns to attain the performance frontier from the heuristics in two network environments: shallow (left) and deep (right) buffers

E.2 Impacts of Different AQMs on Sage

A good learned policy should not depend on any certain assumptions about the environment. Therefore, we investigate the impact of different AQM techniques that can be executed in the network on Sage and other CC schemes. In particular, we setup a 48Mbps, 20ms minimum RTT, and 240KB buffer bottleneck link and use head drop queue (HDrop), tail drop queue (TDrop), PIE [54], BoDe [4], and CoDel [53] as AQM schemes in the network and let flows use different CC schemes and run through this bottleneck link for a minute. We repeat the tests five times and measure the average throughput and delay of them. Fig. 23 shows the results. Sage's performance does not depend on the AQM scheme used, while other schemes are significantly impacted by this change. Again, this can suggest that the policy learned by Sage is a better-performing policy that can scale well to other complex settings.

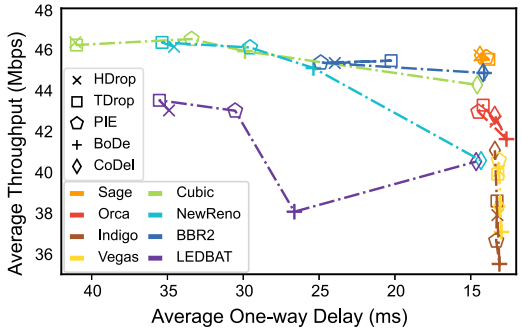


Figure 23: Performance of schemes with diff. AQMs

F SAMPLE RESULTS OF THE LEAGUES

F.1 League of ML-based Schemes

Fig. 24 shows the dynamics of ML-based schemes in some sample environments of Set II (sample small buffer scenario: 120KByte (=80 packets) buffer size, 40ms minimum RTT, & 24Mbps link capacity, and sample large buffer scenario: 1.920MByte (=1280 packets) buffer size, 40ms minimum RTT, 24Mbps link capacity). Sage obtains good throughput and achieves better TCP friendliness in both small buffer and deep buffer environments. Orca and BC-Top show flow starvation in the deep buffer. On the other hand, Aurora is aggressive and cannot coexist with TCP Cubic.

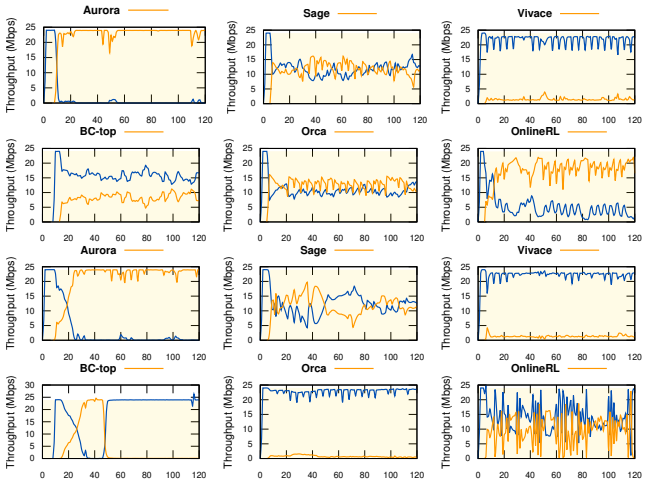


Figure 24: Samples of friendliness aspect of Sage and some other ML-based schemes in small buffer (top two rows) and large buffer (bottom two rows) scenarios

F.2 League of Delay-based Schemes

Fig. 25 presents a sample set of the results from Set II (scenarios are similar to the ones described in section F.1) of some of the schemes in the league of delay-based ones. We observe that Vegas has difficulty getting throughput in the presence of Cubic in both small and large buffers. This also explains Vegas's low ranking in the multi-flow scenarios. Delay-based schemes such as Copa,

C2TCP, and BBR2 have difficulties competing with Cubic flows on large buffer sizes. As expected, these results highlight the well-known challenge of designing delay-based CC schemes to obtain a fair share when coexisting with Cubic flows in different network settings.

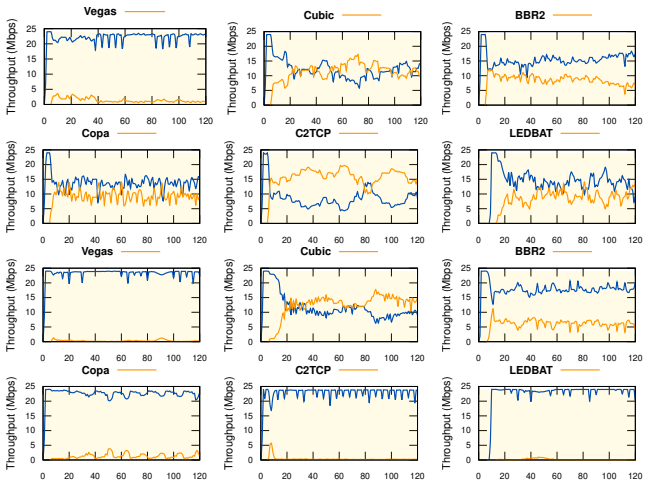


Figure 25: Samples of friendliness aspect of Cubic and a couple of delay-based schemes in small Buffer (top two rows) and large buffer (bottom two rows) scenarios

G INTERNET EXPERIMENTS' SETUP

In addition to our own servers in the US, our Internet experiments consist of 28 servers around the globe on top of Geni [16] and AWS. Table 4 shows the locations of servers. We use different schemes to send 10-second flows between different servers in our experiments. For each source-destination pair, we conduct five trials and report the final average performance. To accurately measure the delay of packets sent in different global timezones, we use NTP servers to synchronize the clocks of client-server pairs before each run. Among all the source-destination pairs, the minimum observed round-trip time spans from 7ms to 237ms.

H SAMPLES FROM EXPERIMENTS ON INTERNET

Samples of some of the results of our Internet experiments detailed in section 6.1 are depicted in Fig. 26.

I SAMPLES OF FAIRNESS ASPECT OF OTHER SCHEMES

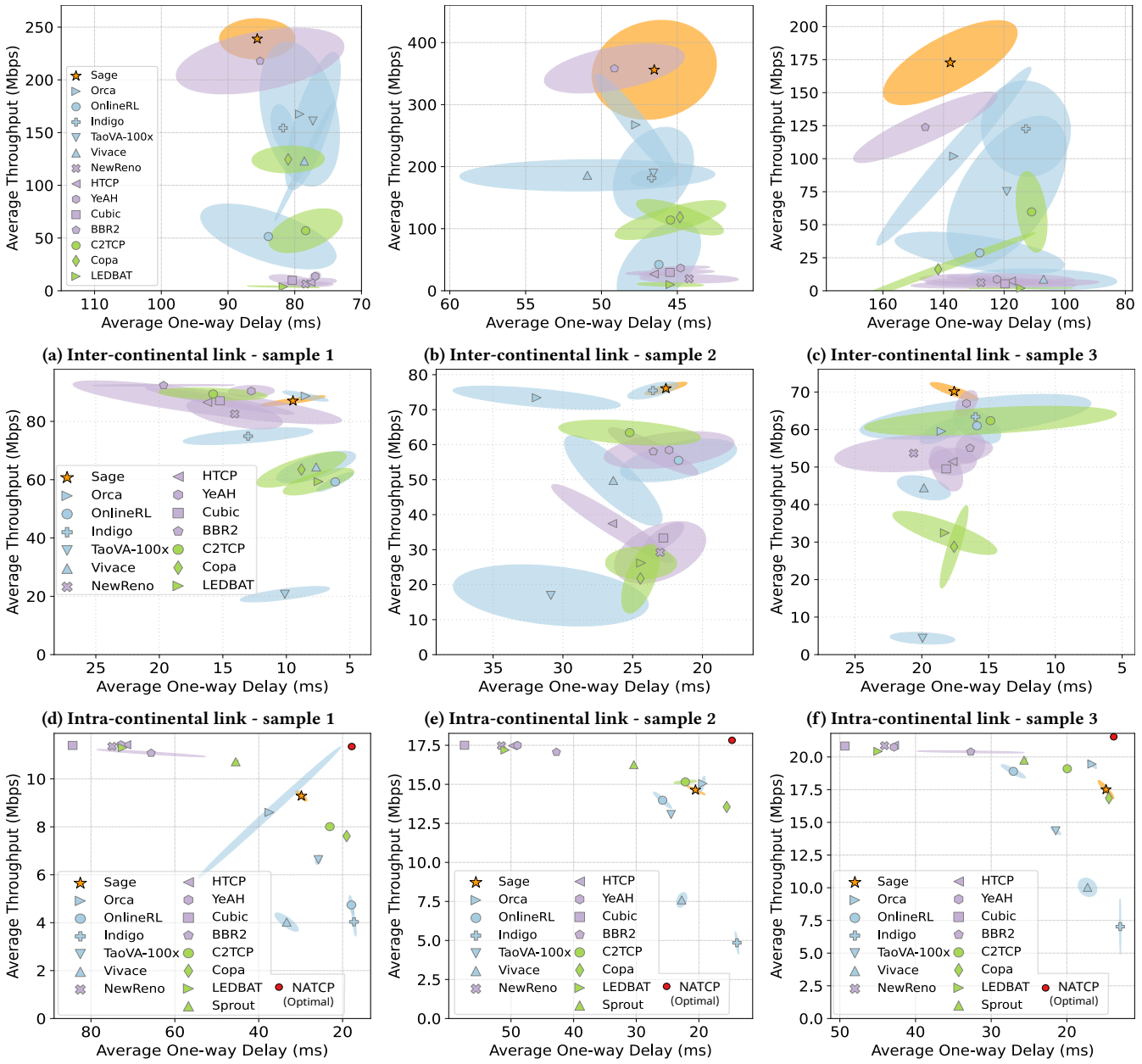
Fig. 27 shows the result of the fairness evaluation described in section 7.7 for different schemes. As mentioned before, fairness is an easier objective to acquire compared to TCP-friendliness due to its definition. Results of Fig 27 confirm that most of the CC schemes achieve the fairness property when competing with flows using the same CC algorithms, though there are still schemes such as Aurora that fail to compete with their own family of protocols.

Table 4: Location of GENI (left) and AWS (right) servers used in our Internet evaluations

Geni Servers	AWS Servers
Tennessee (UTC)	Asia-East (HongKong)
Ohio (OSU)	Asia-Middle East (Bahrain)
Maryland (MAX)	Asia-North East (Osaka)
California (UCSD)	Asia-North East (Tokyo)
Missouri (UMKC)	Asia-South (Mumbai)
Kentucky (UKY)	Asia-South East (Jakarta)
Wisconsin (WISC)	Asia-South East (Singapore)
Ohio (CASE)	Europe-Central (Frankufurt)
Washington (UW)	Europe-South (Milan)
Colorado (CU)	Europe-West (Ireland)
Ohio (MetroDC)	Europe-West (London)
Illinois (UChicago)	Europe-West (Paris)
Missouri (MU)	South America (Sãu Paulo)
California (UCLA)	
Virginia (VT)	

J SAMPLES OF TCP-FRIENDLINESS ASPECT OF OTHER SCHEMES

Fig. 28 shows the result of the evaluation of TCP-friendliness described in section 7.7. Results indicate the more complicated aspect of the TCP-friendliness objective. Competing with a loss-based protocol such as Cubic is hard. The key is not to be very aggressive toward a loss-based scheme (opposite of what Aurora does) and, at the same time, not be very polite and not let a loss-based AIMD scheme take the entire bandwidth (opposite of what Indigo does).



(a) Inter-continental link - sample 1 (b) Inter-continental link - sample 2 (c) Inter-continental link - sample 3

(d) Intra-continental link - sample 1 (e) Intra-continental link - sample 2 (f) Intra-continental link - sample 3

(g) Cellular link 1 (Walking in Times Square) (h) Cellular link 2 (Riding Bus on 6th Avenue) (i) Cellular link 3 (Stationary Pos. in Times Sq.)

Figure 26: Samples of the real-world Internet experiments and emulated cellular networks detailed in section 6.1. We plot the average one-way delay and throughput of different CC schemes. The shaded ellipse represents the $1-\sigma$ variation across runs, while the center marker shows the average value for each scheme.

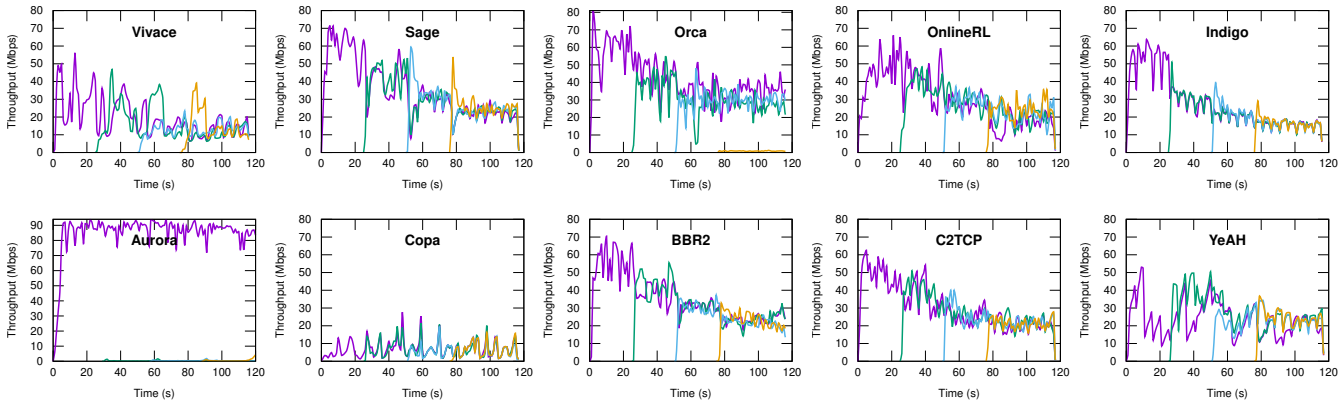


Figure 27: Fairness aspect of Sage, five learning-based schemes, and four heuristic designs

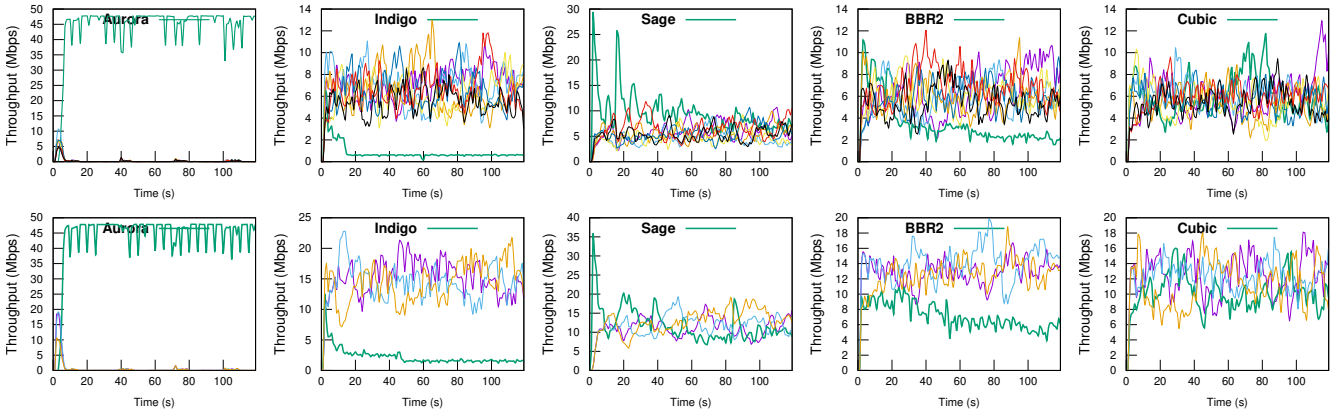


Figure 28: TCP-friendliness aspect of Sage, two learning-based schemes, and two heuristic designs when competing with 7 TCP Cubic flows (top row) and 3 TCP Cubic flows (bottom row) over a 48Mbps link, 40ms mRTT, and BDP buffer