Anole: A Pragmatic Blend of Classic and Learning-Based Algorithms in Congestion Control

Feixue Han, Yike Wang, Yunbo Zhang, Qing Li, *Senior Member, IEEE*, Dayi Zhao, and Yong Jiang, *Member, IEEE*

***Abstract*—In recent years, hybrid congestion control (CC) algorithms that combine rule-based CC and learning-based CC have gained significant attention. They incorporate the fast adaption ability of learning-based CC and the stability of rule-based CC, tending to select the better-performing rate based on the network feedback. However, the practical implementation of such algorithms has revealed primary issues. Specifically, they require both CCs to run alternately, which results in a poorly performing CC continuing to run in the network. Moreover, hybrid CCs cannot converge to the optimal rate when both CCs perform poorly. This paper proposes Anole to address these issues. Anole has three main algorithmic contributions: 1) Anole always selects the better-performing CC, 2) Anole temporarily deprecates the consistently underperforming CC, 3) when both CCs perform poorly, Anole infers the optimal sending rate based on the network feedback. We carry out comprehensive experiments in both emulated and real-world wired networks, as well as in real-world WiFi networks, to assess the performance of Anole. The experiment results demonstrate that Anole achieves approximately 6% higher throughput in real-world links and 34% lower delay in the 48Mbps link compared to the state-of-the-art CC. Anole also exhibits superior performance in adaptability and fair convergence.**

***Index Terms*—Congestion control, machine learning, hybrid algorithm.**

I. Introduction

Congestion control (CC) represents a long-standing and essential area of focus within networking research. Decades of studies on controlling congestion provide us with a plethora of CC algorithms. So far, more than 15 distinct rule-based classic CC algorithms (e.g., CUBIC [1], Vegas [2]) have been integrated into the Linux Kernel [3], [4]. The predictable behavior and minimal overhead characterization make these classic algorithms highly pragmatic [4]. Nevertheless, they exhibit good performance only in certain networks (e.g., CUBIC can cause throughput degradation in high BDP networks [5], and BBR exhibits subpar performance in cellular networks [6], [7], [8]), necessitating extensive manual tuning to adapt to diverse networks.

Given the increasing application requirements and more complex network infrastructures, machine learning has sparked a new wave of revival in CCs [9], [10], [11], [12], [13], [14], [15], which have great potential to adapt to dynamic network conditions with one single control policy [9]. However, it is almost impossible for the training datasets to simulate the complicated real-world Internet dynamics faithfully. Hence, these learning-based CCs occasionally yield significantly inaccurate bandwidth estimation [16]. Besides, the convergence performance of these CCs largely depends on the choice of the utility function (and its parameters), showing limited adaptability [4], e.g., current utility functions usually impose heavy penalties for delay and packet loss, resulting in link underutilization.

To incorporate the advantages and compensate for shortcomings, recent years have witnessed the rise of hybrid CCs that integrate rule-based and learning-based CCs [3], [4], [16], [17], [18], [19]. Hybrid CCs usually adopt a time-division method to run rule-based and learning-based CCs separately, and then select the better-performing one based on the network feedback: Orca [3] employs a learning-based TD3 [20] model to enforce a new congestion window (cwnd) every specific period, which serves as the base cwnd for rule-based CCs. Libra [4] uses a utility function to quantify the performance of different sending rates and choose the one with a higher utility value. Such algorithms intend to complement the adaptability of learning-based CCs and the robustness of rule-based CCs. Nevertheless, this complementary operation may backfire. Considering the limitations of the training set (usually generated through simulations) and the complexity of the actual application scenario, the learning model is likely to face serious out-of-domain generalization challenges, leading to predictions with large deviations. Taking Orca as an example, such erroneous predictions will limit the ability of rule-based CC to explore the correct rate, ultimately

Received 23 July 2024; revised 10 April 2025; accepted 25 April 2025. Date of publication 15 May 2025; date of current version 11 June 2025. This work is supported by the Major Key Project of PCL under Grant PCL2023A06, in part by the Shenzhen R&D Program under Grant KJZD20230923114059020, and in part by Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989. Recommended for acceptance by W. Li. *(Corresponding author: Qing Li.)*

Feixue Han and Yong Jiang are with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China, and also with the Department of Strategic and Advanced Interdisciplinary Research, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: hanfx19@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Yike Wang is with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China (e-mail: wyk23@mails.tsinghua.edu.cn).

Yunbo Zhang is with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China (e-mail: zyb24@mails.tsinghua.edu.cn).

Qing Li and Dayi Zhao are with the Department of Strategic and Advanced Interdisciplinary Research, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: liq@pcl.ac.cn; zhaody@pcl.ac.cn).

Digital Object Identifier 10.1109/TC.2025.3566872

leading to worse performance than using the rule-based CC alone. We reproduce several hybrid CCs and conduct experiments (Section II) to demonstrate that this performance degradation indeed exists and will be more evident when both candidate CCs perform poorly.

To handle situations where one or both CCs perform poorly, the design of a hybrid CC should follow three key principles. (*i*) the hybrid CC's performance must not be inferior to either of the two CC algorithms. (*ii*) except for rate selection, the evaluation of the sending rates should be fully leveraged to infer the right rate. (*iii*) the impact of the long-time poor-performing CC should be minimized. To satisfy these design principles, we propose Anole, a hybrid CC that can correct its behavior when the candidate CCs perform poorly.

Like early attempts of the hybrid CCs [4], [17], Anole employs a utility function to evaluate the performance of the candidate rates (generated from the rule-based CC, learning-based CC, and the historical best) based on the network feedback. If the CC candidates perform well, the rate with the highest utility value is selected to ensure that the best CC is chosen according to the current evaluation criteria. When the candidates show unsatisfactory performance, Anole aims to explore a better rate. Fortunately, the access to multiple rates and their network feedback provide us with additional information to infer the available bandwidth. For instance, the network feedback can tell whether the right rate falls between any two of the candidate rates. Regarding the third principle, under circumstances where the network environment experiences minimal alterations (e.g., link capacity and base delay), CCs that exhibit sustained underperformance struggle to achieve performance enhancement within a short time frame. The continued execution of such CC would result in performance degradation. To achieve this, Anole assigns a confidence score to each CC. Anole reduces the confidence score of the poorly performed CC and will temporarily deprecate the CC when its confidence score falls below a preset threshold. The primary innovations of Anole can be summarized in two key points: First, Anole avoids uncritically relying on any single algorithm; instead, it infers the optimal rate by analyzing feedback from multiple rate assessments. Second, Anole addresses the issue of excessive rate evaluations in current hybrid congestion controls by: 1) extending the control cycle, allowing the optimal rate to be applied for a longer duration, and 2) temporarily suspending the evaluation of algorithms that consistently perform poorly.

Our key contributions in this paper are:

- We analyze the shortage of current hybrid CCs and conduct experiments to demonstrate that a more effective strategy is required to prevent sustained performance degradation when candidate algorithms underperform.
- We design a new three-stage hybrid CC framework. Anole can determine the best decision from the rule-based, learning-based, and previous decisions, and infer a better sending rate when CC candidates show unsatisfactory performance.
- We implement Anole in Linux Kernel and evaluate it with state-of-the-art CCs through Pantheon [21]. Our comprehensive experiments show that compared with other CCs, Anole achieves consistently high performance in various network conditions, especially where one or both CC candidates perform poorly.

II. Background and Motivation

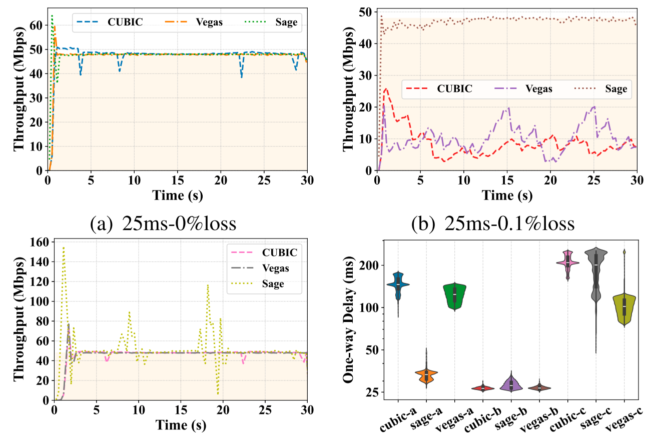
The predictability and low overhead characteristics of rule-based CCs render them prevalent choices for production environments. Nonetheless, they demonstrate good performance exclusively within environments conducive to the specific rule, exhibiting a deficiency in adaptability to alternative networks [3], [4], [9], [22], [23], [24]. Meanwhile, learning-based CCs have demonstrated impressive performance in adapting to different environments, but their robustness has hindered the practical deployment: (*i*) *out-of-domain challenges*: the performance of learning models is heavily dependent on the training set and the duration of training. (*ii*) *trial-and-error mechanism*: reinforcement learning models (commonly used in CC), often yield bandwidth estimates that significantly deviate from the ideal value, resulting in bandwidth overshoot or underutilization [16]. Therefore, an increasing number of researchers have shifted their focus toward Hybrid CCs [3], [4], [16], [18], [19], [25], which aim to combine the benefits of both rule-based and learning-based CC approaches. The fundamental idea of hybrid CC is to select the optimal rate among multiple rate candidates generated by rule-based and learning-based CCs. However, as the following subsection shows, hybrid CCs will encounter several challenges during employment.

A. Rate Deviations in Loss&Long Links

To showcase the challenges that hybrid CCs will face, we run a loss-oriented flow, a delay-oriented flow, and a learning-based flow in different network scenarios. CUBIC [1] and Vegas [2], respectively represent the loss-oriented and delay-oriented CCs. Considering that DRL is widely applied in the CC domain, we choose the state-of-the-art deep reinforcement learning (DRL) algorithm, Sage [18], which is built on the critic regularized regression model of Google (CRR [26]). The experimental scenarios are simulated through the Pantheon [21] platform. The link capacity in all scenarios is 48Mbps.

Fig. 1 shows the performance of these CCs, wherein Fig. 1(a) shows that all flows achieve good throughput performance with 25*ms* one-way-delay and no random loss. CUBIC and Vegas both exhibit high latency (cubic-a and vegas-a in Fig. 1(d)), but for different reasons. CUBIC's high latency is primarily attributed to its buffer-filling characteristic, while Vegas experiences high latency due to the accumulation of queues during the slow start stage. Based on this scenario, we alter the random loss rate and one-way delay of the links to investigate their impact on the CCs' performance.

Scenario 1: link with 25ms one-way-delay and 0.1% random packet loss: When random packet loss is introduced into a network environment, both algorithms are subject to varying degrees of impact. Due to the presence of head-of-line blocking, most rule-based TCP algorithms are highly sensitive to packet loss [1], [2], [27], [28] (BBR [5] is an exception since it relaxes the inflight restriction). As depicted in Fig. 1(b), Vegas only achieves about 20% link utilization. CUBIC imposes additional



(a) 25ms-0%loss (b) 25ms-0.1%loss (c) 50ms-0%loss (d) delays

Fig. 1. Classic and DRL algorithms exhibit different throughput and latency performance in various scenarios. The scenario in (a) is characterized by a 25 ms one-way-delay and no random loss. (b) Adds a 0.1% random packet loss; and (c) alters the one-way-delay to 50 ms.

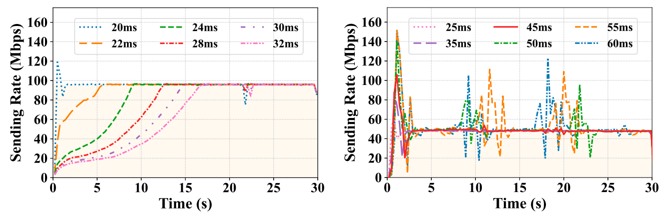
penalties on cwnd, resulting in only a 10% link utilization. With such low link utilization, all flows approach the base one-way delay (25ms). Sage demonstrates superior performance compared to rule-based algorithms. However, it still experiences throughput degradation. Our subsequent experimental findings indicate that an elevated loss rate exacerbates the oscillation of the sending rate.

Scenario 2: link with 50ms one-way delay and no random packet loss: After the base one-way-delay rises to 50ms, Sage shows the most noticeable performance change. The predicted sending rate by the DRL model experiences substantial fluctuations, and the delay rises to a level nearly equivalent to that of CUBIC. Although there are no significant throughput oscillations in CUBIC and Vegas, Fig. 1(d) shows that their delays remain consistently high.

In the examples of the above scenarios, existing hybrid CCs struggle to select the appropriate sending rate. For example, Orca [3] allows the rule-based CC to update the sending rate based on the initial rate, which the DRL model generates at the beginning of each monitoring period. Consequently, in scenario 1, despite the good initial rate provided by the learning-based congestion control algorithm, the rule-based algorithm leads to a rapid drop in the sending rate. In scenario 2, the performance of Orca is primarily driven by the DRL model, which results in fluctuating throughput. Based on Orca, Libra [4] further evaluates each rate and records the historical best rate. However, the optimization for the performance is limited in cases where the optimal rate has never been attained. It is also worth noting that Libra evaluates the rates every three Round Trip Times (RTTs) and cannot evaluate all rates sampled at any given time. As such, the unevaluated rates will not be historical records even if they perform better. In summary, the performance of a hybrid CC is constrained when both types of algorithms exhibit suboptimal performance.

B. Instability of Learning-Based CC

To present the performance of Sage's DRL model under different parameters and network scenarios, we run Sage flows with varying prediction intervals and in simulation environments with different one-way delays. The link capacity is 48Mbps and the base one-way delay is 25ms. The experiment results are shown in Fig. 2.



(a) Different prediction intervals (b) Different one-way delays

Fig. 2. The performance of Sage's DRL model with different prediction intervals and in networks with different one-way delays. The prediction interval varies from 20 ms to 32 ms. The one-way delay varies from 25 ms to 60 ms.

Fig. 2(a) shows how the prediction intervals influence the performance of Sage. Based on our measurements, Sage requires an average time of about 18ms for one inference and shared memory interaction[1]. The default prediction interval in Sage is 20ms and we gradually increase it with a step size of 2ms. The experiment result shows that the convergence time of Sage noticeably increases as the prediction interval increases. For example, an increment of the prediction interval from 20ms to 22ms results in a delay of 5s in the convergence time. Moreover, the flow requires more than 17s to converge to the correct rate when the prediction interval is raised to 32ms. Fig. 2(b) illustrates the sending rate curves under different one-way delays. When the base one-way-delay reaches 45ms, the curve begins to oscillate, and such oscillation intensifies as the one-way-delay increases. After the one-way-delay reaches 60ms, the oscillation range of the rate expands to between 20Mbps and 120Mbps.

Apart from Sage, other learning-based CCs also exhibit varying degrees of instability. For example, PCC Vivace [23] and Indigo [10], tend to impose strict penalties on the growth of RTT and exhibit poor tolerance for network fluctuations. Related experimental results will be presented in section V.

C. Retained Poor CC

The principle of hybrid CC is to run multiple CCs simultaneously and select the sending rate with better performance, which implies that the performance of the unselected rate is often unsatisfactory. However, current hybrid CCs require periodic application of these rates. For instance, Orca sets the sending rate to the rate inferred by the RL model at the beginning of each monitoring period, which is highly dependent on the RL model's performance. Libra employs a cyclic process of running and evaluating the learning-based and rule-based CC. This approach will result in performance degradation if one or both of them perform poorly, especially in stable network environments where the rates are unlikely to change extensively. As a result, these hybrid CCs suffer from persistent performance degradation.

[1]Due to the constraints imposed by Sage's kernel, utilizing GPU during the model inference process is not feasible.

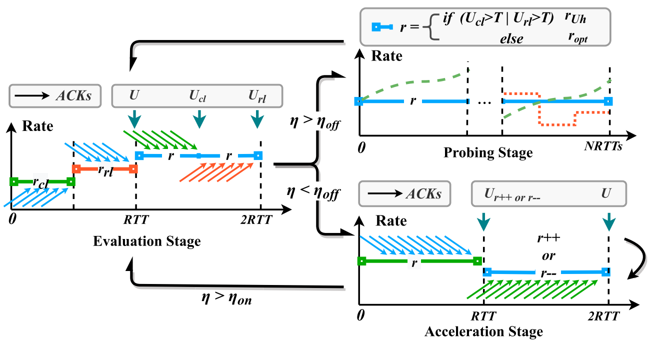


Fig. 3. The finite state machine of Anole. Anole transfers among the evaluation stage, the exploration stage, and the acceleration stage.

In practice, a CC that exhibits rate deviation over several consecutive RTTs can be deemed unsuitable for the current network environment (e.g., the random packet loss in sample 2). In such cases, the CC's performance may not significantly improve until the network environment changes. Specifically, in the presence of random packet loss or when the DRL model encounters an out-of-domain generalization challenge, the CC's sending rate will not converge to the available bandwidth over time. Hence, when a CC consistently exhibits low performance, we should consider deprecating it for a period and re-evaluating its effectiveness after the network environment changes. These observations highlight the need for more effective approaches in hybrid CC to adapt to the dynamic and complex network environment.

III. Design

In general, Anole utilizes a utility function to evaluate the performance of three sending rates, which calculates the utility values for the rate of the rule-based CC (r_{cl}), the rate of the learning-based CC (r_{rl}), and the rate applied in the probing stage (r). Our key innovation lies in 1) *inferring the optimal sending rate through these rates and their corresponding utility values.* 2) *temporarily deprecate the consistently underperforming algorithms to reduce performance degradation.*

A. Overview of the Finite State Machine

Fig. 3 shows the finite state machine of Anole, which is composed of the evaluation stage, the probing stage, and the acceleration stage. Anole starts with a slow start phase to detect the available bandwidth swiftly and then enters the finite state machine.

Anole starts from the evaluation stage (Section III-B), where a utility function assesses all the rates. The utility function considers the throughput, queue length, and queue dynamics, aiming at achieving high throughput and low latency simultaneously. After obtaining the utility values of r, r_{cl}, and r_{rl} (U, U_{cl}, U_{rl}) at 1RTT, 1.5RTT, and 2RTT, Anole checks if there exists an algorithm (rule-based or learning-based CC) with consistently poor performance. If no, Anole enters the probing stage (Section III-C). The sending rate in the probing stage is denoted as r. During the probing stage, Anole determines whether to use the rate with the maximum utility value ($r = r_{U_{max}}$) or to deduce an optimal rate ($r = r_{opt}$, *the optimal rate refers to the fair bandwidth share in the current network scenario and it equals to the link capacity if there only exists a single flow*) for data transmission in the next N RTTs. Otherwise, Anole enters the acceleration stage (in Section III-D). In this stage, the underperforming CC will be temporarily deprecated, which can reduce the computational overhead, minimize the performance loss, and accelerate the convergence. After a few RTTs, we will re-evaluate this deprecated CC to determine whether it can work properly in the current network scenario.

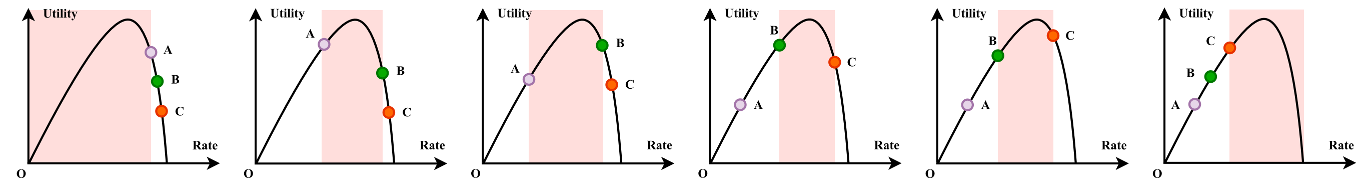
Algorithm 1: Anole Algorithm.

```
1  foreach control cycle $t$ do
2      // evaluation stage
3      sending traffic with rate $r$ for 2RTT;
4      for time in next 1/2 RTT do
5          $r = min(r_{cl}, r_{rl})$
6      for time in next 1/2 RTT do
7          $r = max(r_{cl}, r_{rl})$
8      calculate the utility value $U_{prev}$
9      for time in next RTT do
10         $r = r_{prev}$
11     calculate the utility value $U_{cl}$ and $U_{rl}$
12     update $\eta_{cl}, \eta_{rl}$ according to Algo. 2
13     // decide the initial rate in the next stage
14     if $U_{cl} > T$ || $U_{rl} > T$ then
15         $r = argmax_{r_{cl}, r_{rl}}(U_{cl}, U_{rl})$
16     else
17         computing $r$ according to Eq. 7
18     // probing stage
19     if $\eta_{cl} > \eta_{off}$ & $\eta_{rl} > \eta_{off}$ then
20         sending traffic with rate $r$ for N-1 RTTs
21     // acceleration stage
22     else
23         while $\eta_{cl} < \eta_{off}$ || $\eta_{rl} < \eta_{off}$ do
24             perform AIAD based on rate $r$
25     // obtain the $r_{rl}$ for next evaluation stage
26     sending traffic with rl-algorithm for an RTT
```

B. Evaluation Stage

Anole initializes with the slow start phase to quickly reach the available bandwidth. During the slow start process, the learning-based CC runs in the background, allowing Anole to directly enter the evaluation stage after quitting the slow start phase. The value of r is initialized to $(r_{cl} + r_{rl})/2$ when entering the evaluation stage for the first time. In subsequent control cycles, r is assigned the sending rate applied in the previous probing stage. Algorithm 1 shows the workflow of Anole in each control cycle.

In the evaluation stage, rates generated by rule-based (r_{cl}) and learning-based CC (r_{rl}) run in ascending order for 0.5 RTT, respectively (as shown in lines 4-7 in Algorithm 1). Running at



(a) $r_{U_h} = r_a,\ \Delta D_a > 0$ (b) $r_{U_h} = r_a,\ \Delta D_a \le 0$ (c) $r_{U_h} = r_b,\ \Delta D_b > 0$ (d) $r_{U_h} = r_b,\ \Delta D_b \le 0$ (e) $r_{U_h} = r_c,\ \Delta D_c > 0$ (f) $r_{U_h} = r_c,\ \Delta D_c \le 0$

Fig. 4. Distributions of r_{cl}, r_{rl}, and r (denoted as r_a, r_b, and r_c in ascending order) and the location of optimal rate.

a lower rate initially helps to reduce the side effect of the bottleneck queue caused by the first-running CC. During this RTT period, Anole continuously receives ACK feedback (the blue arrows) from the last probing/acceleration stage. Therefore, Anole can calculate the U_{prev} the previous rate just after the first RTT (line 8). Similarly, in 1-1.5 and 1.5-2 RTTs, Anole receives the feedback of r_{cl} and r_{rl} (the green and orange arrows) and calculates their utility values at 1.5 and 2RTTs (line 11). Note that if r is not a calculated r_{opt}, there is no need to re-compute the utility value of r, as its utility (U_{cl} or U_{rl} in the last evaluation stage) has already been obtained in the previous round. At 2 RTT, Anole has completed data collection and evaluation (with the utility function) for all rates.

Utility function: By default, Anole leverages a utility function as follows (we use x in the utility function to represent any type of rate for convenience):

$$u(x) = \alpha x^t - \beta x \cdot max\left\{\frac{d(RTT)}{dt}, 0\right\} - \lambda x \cdot \frac{RTT}{RTT_{min}} I_Q \quad (1)$$

where $0 < t < 1$, x is the sending rate in Anole, the hyper-parameters $\alpha,\ \beta,\ \lambda > 0$, I_Q is an indicator function defined as follows:

$$I_Q = \begin{cases} 0 & \dfrac{RTT}{RTT_{min}} \le 1 + \mu \\ 1 & \dfrac{RTT}{RTT_{min}} > 1 + \mu \end{cases}, \qquad \mu > 0 \quad (2)$$

Our utility function consists of three components. ***The first term*** in Equation 1 encourages a higher sending rate, which ensures that the utility function is a convex function. ***The second term*** penalizes the RTT growth. The minimum value for the delay gradient is set to 0, as a decrease in RTT indicates that the current sending rate is lower than the optimal rate and should not be considered as a reason for an increase in the utility value. ***The third term*** penalized the in-network queue length. Considering the need to fully utilize the network bandwidth, the in-network queue cannot be completely eliminated. Therefore, in Equation 2, we employ μ to control the tolerable queue length. When $\frac{RTT}{RTT_{min}}$ is less than $1 + \mu$, it implies that the in-network queue length is within an acceptable range and will not impose any penalty on the utility value.

This utility function has two characteristics: 1) It does not penalize packet loss, because the constraints imposed on the gradient and absolute value of RTTs are deemed sufficient to prevent individual flows from causing network congestion. With these constraints, the packet loss events are highly likely to result from other traffic or random packet loss (e.g., due to link failures). Besides, this ensures the convergence of both the sending rate and in-network queue length to an optimal state, preventing scenarios where only one side converges (e.g., the sending rate equals the available bandwidth with a persistent deep queue at the bottleneck). 2 All the penalty terms are multiplied by the sending rate x, because the penalties applied to the utility value should correspond to the contribution of each flow to the network congestion.

C. Probing Stage

Anole has to determine the sending rate in the probing stage. Due to the inherent inaccuracies in inferring the optimal rate and the associated computational overhead, Anole prefers to use either the rule-based or the learning-based CC and only estimates the optimal rate when both CCs perform poorly. Therefore, Anole utilizes a threshold T (T is a multiple of U_{max}) to check whether there is an acceptable rate. If either the utility value of r_{cl} or r_{rl} (or both) is greater than T, Anole adopts the rate with the higher utility value (r_{U_h}) (lines 14-15 in Algorithm 1). Otherwise, Anole chooses to infer the optimal sending rate based on the information collected from the two different rates (lines 16-17). Next, we will introduce **how to infer the optimal rate.**

Now we have gathered three rates $r_{rl},\ r_{cl},\ r$ and their utility values $u_{rl},\ u_{cl},\ u$. Then Anole infers the optimal rate based on these values. If we arrange $r_{rl},\ r_{cl},\ r$ in ascending order as $r_a,\ r_b,\ r_c$, the optimal rate can occur at four possible locations. (*i*) Situation I: In Fig. 4(a), $r_a,\ r_b,\ r_c$ are all higher than the optimal rate (the highest point of the curve). r_a possesses the largest utility value and the measured delay gradient ($\Delta D = d(RTT)/dt$) is positive. (*ii*) Situation II: the optimal rate locates between r_a and r_b. In Fig. 4(b), r_a possesses the largest utility value and ΔD is non-positive. In Fig. 4(c), r_b holds the largest utility value and ΔD is positive. (*iii*) Situation III: Only r_c is higher than the optimal rate. Fig. 4(d) and 4(e) illustrate this scenario, where r_b and r_c holds the largest utility value, respectively. (*iv*) Situation IV: $r_a,\ r_b,\ r_c$ are all lower than the optimal rate. In Fig. 4(f), r_c possesses the largest utility value and ΔD is non-positive.

For situation II and situation III, if the difference between the two rates closest to the optimal rate is small, we directly select the rate with the highest utility value. Otherwise, we infer the optimal rate to avoid consistently choosing rates that deviate significantly from the optimal value:

In situations I and II, the optimal rate can be directly inferred. Expressing Eq1 in a piece-wise manner, we have:

$$u(x) = \begin{cases} \alpha x^t & x + R \leq C \\ \alpha x^t - \beta x \cdot \dfrac{d(RTT)}{dt} - \lambda x \cdot \dfrac{RTT}{RTT_{min}} \cdot I_Q & x + R > C \end{cases} \quad (3)$$

where R is the aggregated rate of all other flows. When $x + R > C$, we can represent the delay gradient with x, R, and C:

$$\Delta D = \frac{d(RTT)}{dt} = \frac{x + R - C}{C} \quad (4)$$

By substituting Equation 4 into Equation 3, the resulting equation contains two unknown variables, namely the link capacity C and the sum of other rates R. If two rates are higher than the optimal rate (Fig. 4(a), 4(b), 4(c)), it becomes possible to deduce the values of C and R, thereby facilitating a more refined analysis of the system. The location where the utility function reaches its maximum point is the intersection of two segments, that is:

$$\alpha x^t = \alpha x^t - \beta x \cdot \frac{d(RTT)}{dt} - \lambda x \cdot \frac{RTT}{RTT_{min}} \cdot I_Q \quad (5)$$

specifically, when the in-network queue length is within the threshold ($I_Q = 0$), x should satisfy:

$$x + R = C \quad (6)$$

Considering that other flows also relinquish network bandwidth after receiving feedback, we should assume that R decreases proportionally to r to ensure sufficient utilization and fair allocation of the bandwidth, we have:

$$x' = C - R \cdot \frac{C}{R + x} \quad (7)$$

When $I_Q = 1$, it indicates the presence of a congested queue in the network. However, this does not imply the current sending rate is incorrect. Rather, it necessitates a temporary reduction in the sending rate to alleviate the in-network congestion. According to the recorded RTT, we can estimate the total volume of the queued packets at the bottleneck as:

$$\left(\frac{RTT}{RTT_{min}} - 1\right) \times BDP = (RTT - RTT_{min}) \times C \quad (8)$$

The contribution of the current flow to the queue can be quantified as $(RTT - RTT_{min}) \times x$. Then we choose to send data at a rate of:

$$max\left(\frac{RTT_{min}}{RTT} \times x,\ 0.5x\right) \quad (9)$$

for an RTT to consume these accumulated packets. To ensure the sending rate does not become excessively low, a minimum rate of $0.5x$ is enforced.

In situation III, only one rate lies in the second segment of Equation 3, making it impossible to simultaneously deduce C and R. However, if situations I and II have occurred previously, we consider employing the historical value of C, as the link capacity is unlikely to change frequently[2]. With this historical value C, we can deduce the value of R and then calculate the optimal rate through Equation 7. If neither situation I nor II has ever occurred, we take the median of the two rates closest to the optimal rate (r_b and r_c) as the optimal rate.

In the situation IV, In this scenario, all rates are located in the first segment of Equation 3, which means that all rates are lower than the optimal one. Therefore, we consider a growth on top of r_c. To minimize network oscillations, we believe that the growth step should decrease as the rate approaches the optimal point. The first segment of Equation 3 is a convex function since $0 < t < 1$, which allows us to apply gradient ascent to rate control:

$$r = r_c + \gamma \cdot \frac{d(u(r))}{dr},\ \ \gamma > 0 \quad (10)$$

where γ is the confidence amplifier (the concept of the confidence amplifier has been employed in multiple previous algorithms [22], [23], [29]). When the sender repeatedly decides to increase the rate, the confidence amplifier is increased.

The probing stage will last for N RTTs, where $N \geq 2$. During this period, rule-based algorithms would continue to operate based on rate r as (i) their computational overhead is minimal, (ii) a single RTT is usually insufficient to induce a significant change in the rate of rule-based CCs. In contrast, learning-based algorithms would be executed only in the final few (one) RTTs of the probing stage to conserve computational resources. The probing stage allows the algorithm to make occasional errors in network judgment, and only when a CC shows consistent performance loss does it enter the acceleration stage.

D. Acceleration Stage

A particular algorithm may exhibit persistent suboptimal performance. For learning-based CCs, due to the constraints of the training data and the complexity of real-world application scenarios, the learning model is susceptible to significant out-of-domain generalization challenges, which can result in predictions with substantial deviations (e.g., the training dataset ranges from 1Mbps to 40Mbps, while the test link has a capacity of 200Mbps). Scaling the dataset can potentially mitigate the issue, but the heterogeneity of network conditions poses a challenge to the complete coverage of the training data. Meanwhile, rule-based CCs fail to adapt to specific environments. For instance, CUBIC exhibits significant throughput loss in the presence of random packet loss, while BBR experiences sustained high latency in wireless networks. In conclusion, both rule-based and learning-based CCs can seriously deviate from the optimal rate. In the face of these scenarios, persisting with the evaluation and probing stage would result in a squandering of network and computational resources. Therefore, Anole specifically designs a stage for this case: acceleration stage.

Intuitively, CCs that continuously exhibit poor performance should be deprecated. To this end, we propose assigning a confidence score (denoted as η) to each CC. All CCs have the same

[2]Considering the severe bandwidth fluctuations of wireless networks, we will not use historical values in wireless environments.

Algorithm 2: State Transition
state: if stage = deprecate, Anole enters the acceleration stage; if stage = activate, Anole enters the evaluation and probing stages.

```
1  function state_of_CC()
2      if state == activate then
3          $\eta = \min(1, \eta \cdot y)$
4          if $\eta < \eta_{off}$ then
5              $\eta = \eta_{off}$
6              state = deprecate
7      else
8          $\eta = \eta + \Delta\eta$
9          if $\eta > \eta_{on}$ then
10             state = activate
11     return state
```

initial confidence score, which is updated at the end of each evaluation stage (line 12 in Algorithm 1). When a CC's rate shows a long-term deviation from the optimal rate (its utility value is significantly lower than the maximum utility value), we consider it to perform poorly in the current environment, and its confidence score should be reduced. When one or both CCs' confidence score falls behind the threshold η_{off}, Anole enters the acceleration stage (lines 23-24). Then we will introduce how Anole updates the confidence score for each algorithm and adjusts the rates in the acceleration stage.

Confidence score update: The performance of the current CC is assessed based on the ratio y between the current utility value and the optimal utility value:

$$y = \frac{u_r}{u_{max}} + \theta, \qquad \theta > 0 \tag{11}$$

where θ represents the allowable deviation of the utility value. When $y > 1$, the confidence score increases; otherwise, it decreases. The state conversion of each CC is shown in Algorithm 2. The maximum value of η is set to 1. At the end of each evaluation stage, we multiply η by y to adjust it (line 3). If the observed $y > 1$, η increases, otherwise, η is reduced multiplicatively. As η decreases below the deprecate threshold η_{off}, η is set to η_{off}, and the state of the CC is marked as deprecate (line 6). η increases by $\Delta\eta$ for each 2RTTs until it grows to η_{on} (line 8), after which the state is set to activate. Next, based on the current sending rate, we will run the rule-based and learning-based CC for one RTT, respectively (to provide the rate for evaluation), and re-enter the evaluation stage. This approach guarantees equal periods of deprecation for each algorithm and provides equal opportunities for reactivation.

Rate adjustment: The initial rate selection in the acceleration stage is similar to that in the probing stage. If there is a CC in the 'activate' state, the rate of the activated CC is adopted as the initial rate. Otherwise, the inferred optimal rate is chosen as the initial rate. During the acceleration stage, Anole performs an additive increase/decrease operation every two RTTs based on the value of the current smoothed RTT. Specifically, when $(RTT/RTT_{min}) > 1 + \mu$, Anole performs an additive decrease, otherwise, it performs an additive increase. We use a velocity parameter v to accelerate Anole's rate adjustment. v is initialized as 1. If Anole performs consecutive increase/decrease operations, v increases exponentially by a factor of 2. Then Anole adjust the cwnd by v packets each time. Once the adjustment direction changes, v is reset to 1.

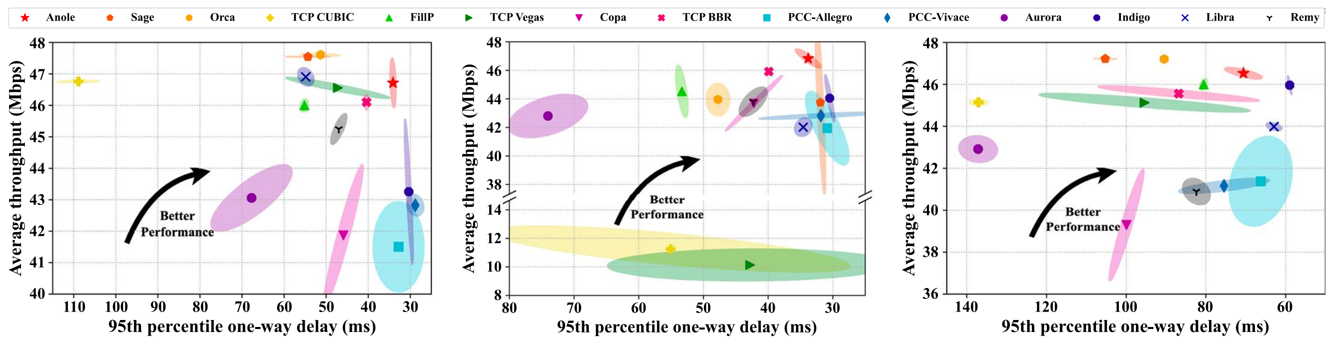
IV. Implementation

Anole only requires modifications to the sender and does not need any additional cooperation from the receiver. Anole supports the use of any rule-based CC, including but not limited to the loss-based CUBIC [1], delay-based Vegas [2], and learning-based CC, including but not limited to Indigo [10], Aurora [9], and Sage [18]. In our experiments, we choose CUBIC as the rule-based CC since CUBIC is the default CC in the current Linux system. Sage is chosen as the learning-based CC because 1) Sage is the current state-of-the-art DRL model. 2) the deployment of Sage does not require modifications to the kernel. At the same time, we also demonstrate that even the most advanced learning model shows performance degradation in certain network environments.

We use Pantheon [21], an independent platform that serves as a training ground for research on congestion control, to evaluate the performance of Anole. The RL-based CC in Anole is implemented in userspace and the rule-based CC is implemented in Linux kernel. We modify the Linux kernel to enable flows to send data at a fixed rate over a while. To evaluate Orca [3], Sage [18], we deploy the Kernel of Sage (compatible with Orca) on a server and integrate them into Pantheon. We reproduced Libra [4] based on the ns3 simulation code provided by the authors and similarly deployed it in the kernel. Due to the fact that Spine [17] has not open-sourced the code or the dataset, we face challenges in reproducing Spine's learning models.

Our experiments are conducted on servers with Intel (R) Xeon (R) CPU and RTX 3090 GPU. Due to the deployment of different kernels (version 4.19 for Sage and version 5.15 for Anole), Anole and Sage are evaluated on different servers. The server has 48 CPU cores and a total memory of 252 GB. We evaluate Anole through simulations with Pantheon [21]. During our experiments, we set the parameters of Anole as follows: $t = 0.9$, $\alpha = 1$, $\beta = 900$, and $\gamma = 11$. The thresholds of confidence score, η_{on} and η_{off}, were set to 0.6 and 0.8, respectively. T in the probing stage is set to 0.85. The probing stage lasted for 3 RTTs.

We referenced the parameter settings in several previous works that utilized utility functions [4], [22], [23] and tested multiple values around them. After setting α to 0.9, we conduct experiments to observe the performance of Anole with β ranging from 600 to 1200. From 600 to 900, we see periodic delay inflation with a peak of ten times the base RTT. The delay inflation will subsequently decrease with the penalties of the absolute RTT, while the delay fluctuation cannot be eliminated. From 900 to 1200, the delay gradient is excessively punished, leading to throughput loss. Therefore, we suggest a β around 900. When γ is set to 11, a single flow can achieve a 95% link utilization. Theoretically, a larger γ allows for a faster response to the excessive delay caused by a higher rate. However, as γ increases, the flow's resistance to burst traffic decreases, which can also lead to



(a) 25ms-2BDP (b) 25ms-0.1%-2BDP (c) 50ms-2BDP

Fig. 5. Throughput and 95th delay for all the compared CCs in normal link (a), loss link (b), and long link (c).

throughput losses. η_{off} reflects the tolerance for a consistently poorly performed algorithm. The higher the value of η_{off}, the more easily an algorithm can be disabled. The distance between η_{on} and η_{off} determines the minimum time an algorithm can be disabled: $(\eta_{\text{on}} - \eta_{\text{off}})/0.1$*RTT. T represents the acceptable distance between the utility value of the current rate and U_{max}. Unlike η_{off}, T indicates the tolerance of a single poor performance. A larger T suggests a higher probability of inferring the optimal rate, ensuring small fluctuations in the algorithm's performance. A smaller T indicates fewer inference executions, which can reduce the computation overhead.

V. Evaluation

In this section, we compare the performance of Anole with the Linux implementation of several CCs: CUBIC [1], BBR [5], Vegas [5], and user-space implementation of several CCs: Sage [18], Orca [3], Fillp [30], Copa [29], PCC [22], Vivace [23], Aurora [9], Indigo [10], Libra [4], Remy [11]. Each algorithm was tested with its default parameter configuration. Each experimental result represents the average of more than 10 runs, with each run lasting for 30 seconds.

A. High Performance in Simulated Networks

To evaluate Anole's throughput and delay performance, we repeat the network scenarios in Section II-A and run all the compared CCs 10 times. The link capacity in all scenarios is set to 48Mbps and the buffer size is set to 2× BDP. Fig. 5 shows the performance of all the compared CCs.

In Fig. 5(a), all CCs run on the link with a 25ms one-way-delay. Under this scenario, Sage simultaneously achieves the goal of high throughput (about 47Mbps) and low latency (approximately 40ms). CUBIC also attains high throughput. However, its latency surpassed 120ms due to the buffer-filling characteristic. In this context, Anole primarily preferred the rate inferred by Sage. Given the strict limitation on the gradient and absolute value of RTT, Anole will infer an optimal rate if both CUBIC and Sage perform poorly in terms of delay, which brings a slightly lower latency and throughput than Sage. As described in Section II-B, learning-based CCs (including Aurora, Remy, PCC, Vivace, and Copa) tend to penalize the RTT growth and lead to link under-utilization excessively.

TABLE I
The Throughput and Delay of BBR and Anole With Different Flow Numbers in a 96 Mbps Link

2*Method	1 flow		3 flows		5 flows		10 flows	
	thpt (Mbps)	95th delay (ms)	thpt (Mbps)	95th delay (ms)	thpt (Mbps)	95th delay (ms)	thpt (Mbps)	95th delay (ms)
Anole	64	90.62	78	92.07	86	93.72	93	94.03
BBR	72	91.33	108	93.88	129	94.63	142	94.77

Random packet loss is introduced into the network scenario in Fig. 5(b). CUBIC and Vegas are severely affected, with only a 0.1% loss rate, causing their link utilization to be under 20%. CUBIC and Vegas' latency instability is attributed to the time required to digest the queue buildup during the slow start. The throughput of Orca, Remy, Aurora, and Copa only shows a slight decrease, but their latency shows a certain fluctuation. In this scenario, though Sage maintains an overall good performance, its prediction results show unavoidable fluctuations (usually lower than the available bandwidth). In Fig. 5(a), Anole primarily chooses the rate of Sage but also counteracts Sage's oscillations with the inferred optimal rate. Therefore, Anole presents high throughput with a small variance.

As shown in Fig. 5(c), when the one-way delay rises to 50ms, the delay-based CCs experience varying degrees of oscillation. Sage's rate prediction bias in long links (as shown in Section II-A) leads to a drastic increase in the latency. In this context, both CUBIC and Sage's performance are unsatisfactory. Therefore, aside from the probing stage, Anole primarily utilized the inferred optimal rate, ensuring low latency even when Sage and Orca do not perform well.

Table I presents the throughput and latency for different numbers of Anole flows over a 96Mbps link, with a base RTT of 50ms and a buffer of 2BDP. We also include BBR's performance for comparison because we observe that BBR's latency rises significantly as the flow number increases. As the number of flows increases from 1 to 10, both Anole and BBR show an approximate 3% increase in link utilization. However, Anole's RTT increased by only 30 ms, whereas BBR's escalated by 70 ms. This difference is attributed to BBR's reliance on the measured maximum sending rate. In

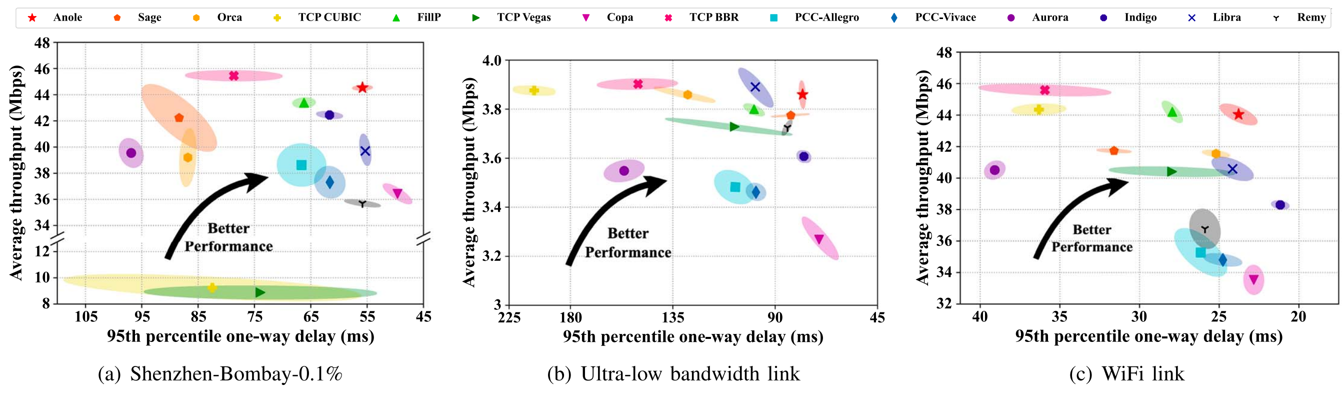


Fig. 6. Throughput and 95th delay for all the compared CCs in real-word links.

contrast, Anole consistently adjusts towards the optimal rate defined by its utility function.

B. High Performance in Real-World Links

To evaluate the performance of various algorithms in complex real-world scenarios, we established two distinct links: 1) A link between Shenzhen and Mumbai, characterized by a baseline RTT of approximately 90 ms, as determined by multiple RTT measurements. The link capacity is constrained to 48 Mbps, and we introduced a random packet loss rate of 0.1% using tc. This link can be classified as a long-distance link with inherent packet loss. 2) A WiFi link, where we equipped the servers with wireless NICs to assess the performance of congestion control algorithms under wireless conditions. The latency in the wireless network exhibits continuous fluctuations, with measurements indicating a baseline RTT of less than 40 ms. 3) An ultra-low bandwidth link between Shenzhen and Mumbai. The link capacity is constrained to 4Mbps, and the baseline RTT is around 90ms (45ms one-way delay).

Fig. 6 presents the experimental results obtained from real-world links. In Fig. 6(a), the presence of packet loss leads to persistently low throughput for CUBIC and Vegas, which also impacts the throughput of Orca and Libra. Although Anole is also affected by CUBIC, its lower evaluation frequency allows it to maintain relatively high throughput levels. Compared to the simulated links, both Sage and Orca exhibit increased performance fluctuations in the real link. PCC, Vivace, and Copa all demonstrate low link utilization in such a long link. Aurora and Remy display consistent performance with their behavior in simulated scenarios: Aurora consistently achieves higher latencies, while Remy maintains low link utilization. Fig. 6(b) shows the performance of CCs under an ultra-low latency link. In this link, all CCs achieve a high bandwidth utilization (more than 80%). Consuming queued packets takes longer on low-bandwidth links. Therefore, the measurement error of the link bandwidth by BBR has a more significant impact on latency. Meanwhile, the impact of running CUBIC on Orca's latency is also more pronounced. Anole ensures stable high bandwidth and low latency in low bandwidth scenarios.

Fig. 6(c) shows the performance of CCs in a real WiFi network. All learning-based algorithms experience varying degrees of throughput degradation due to their difficulty in adapting to fluctuating network capacity. Conversely, classic CCs exhibit performance advantages in throughput. Due to the high RTT variability in the WiFi network, Copa demonstrates the lowest throughput. Notably, Anole frequently enters the acceleration stage because the learning-based CC usually makes bad decisions, thereby enhancing its stability.

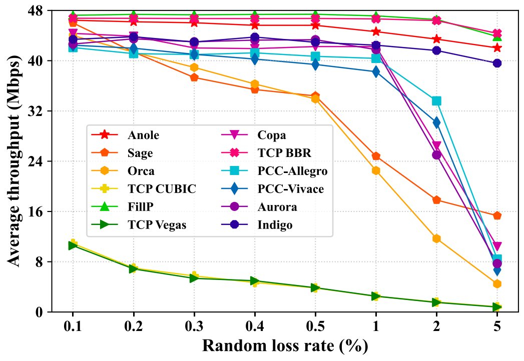


Fig. 7. The throughput of CCs under different random loss rates. The link bandwidth is 48Mbps and the loss rate ranges from 0.1% to 5%.

C. Resistance to Random Loss Rates

In this section, we demonstrate that despite using CUBIC as the rule-based CC, Anole can effectively withstand random packet loss. The loss rate varies from 0.1%-5%. In Fig. 7, BBR, Fillp, and Indigo exhibit relatively stable performance in the face of packet loss, where BBR and Fillp achieve the highest throughput, maintaining over 90% link utilization even at a 5% loss rate. CUBIC and Vegas already exhibit link utilization below 20% at a loss rate of 0.1%, and this throughput loss further exacerbates as the loss rate increases. Sage and Orca experience a drastic decrease in throughput after the packet loss rate reaches 0.5%.

When the loss rate is below 0.5%, Sage experiences only limited affection while CUBIC remains mostly deprecated due to its low throughput. Therefore, Anole's throughput only experiences a slight decrease. However, due to the presence of CUBIC, Anole's throughput inevitably falls slightly below that of BBR. When the loss rate exceeds 0.5%, both Sage and CUBIC exhibit low link utilization, indicating that Anole requires a longer time to explore rates with higher utilization. Although we observe a more pronounced throughput decrease in Anole, Anole still

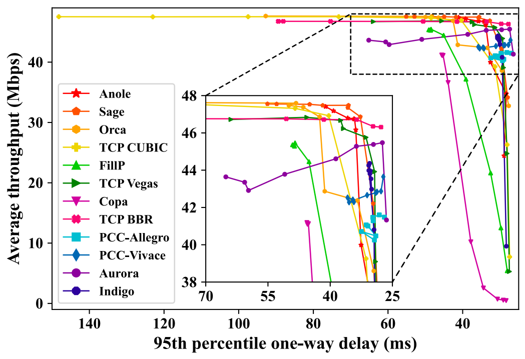


Fig. 8. The throughput and 95th delay of CCs under different buffer sizes. Each line in the figure starts with a larger marker indicating the start point and the buffer size of other points increases in a zigzag order. The buffer sizes are [0.04, 0.08, 0.16, 0.32, 0.5, 1, 1.5, 2, 2.5] × BDP.

surpasses other CCs. These findings demonstrate the robustness od Anole when facing random packet loss.

D. Impact of Buffer Sizes

In this section, we evaluate the impact of buffer size on different CCs. As buffer size has a more pronounced effect on both throughput and delay, in Fig. 8, the $x-axis$ is 95th delay and the $y-axis$ is throughput. The buffer sizes are in [0.04, 0.08, 0.16, 0.32, 0.5, 1, 1.5, 2, 2.5] x BDP. The performance of CCs at the buffer size of 0.04xBDP is marked with a larger point and the rest of the data is read in the order of zigzag lines.

The experiment results show that the performance of PCC-Vivace, PCC-Allegro, and Indigo is almost unaffected by buffer size. The delay of Aurora, BBR, and Vegas significantly increases with the growth of buffer size. At a buffer size of 2.5xBDP, CUBIC's delay is three times higher than that at 0.5xBDP. In the same scenario, Orca and Anole experience a delay increase of approximately two times. Sage and Anole also experience a slight increase in delay, but when the buffer size rises to 2.5xBDP, Anole achieves an approximately 34% lower delay than Sage (38ms and 60ms, respectively). A low buffer size makes it difficult for some CCs to fully utilize the link. Despite Fillp using BBR as its congestion control algorithm, buffer size has a much greater impact on its performance than on BBR, due to Fillp's implementation of ACK compression. Copa requires a certain in-network queue to estimate the optimal rate, which makes it more sensitive to the buffer size. Besides, due to the limitation of inflights, CUBIC and Vegas are also highly sensitive to buffer size.

E. Convergence and Fairness

To demonstrate the performance of CCs in terms of fairness and convergence, we start four flows at 4s intervals on a 48Mbps link and observe how their throughput changes. Flows are considered converged when the sending rate of a flow fluctuates around the fair bandwidth share.

The experiment results are shown in Fig. 9. Except for Anole, Fig. 9(b) to 9(d) show the performance of online-learning CC. PCC only penalizes loss and adopts a MIMD method, which results in a non-convergent state. Based on PCC, Vivace optimizes the utility function to a convex function, improving PCC's convergence. Although Vivace requires quite a long time to converge (after 25s), the sending rates of all four flows are approaching the fair point. Copa determines a common target rate for all flows based on the queuing delay, reaching a fast and stable convergence.

Fig. 9(e) to 9(h) show the performance of DRL-based CC. DRL-based CCs cannot observe the state of the background flows during the training process and infer rates in a black-box mode, making fairness a strong weakness of them. Among DRL-based CCs, Orca shows a relatively good convergence since Orca runs CUBIC based on the rate inferred by the model. Aurora slowly moves towards the fair point and shows convergence after 28s. Indigo has been in a state of continuous and significant oscillation and has not converged (flow 4 has a divergent trend). Sage flows' sending rates change smoothly, but the rates of each flow converge erroneously far from the fair bandwidth share.

Fig. 9(i) to 9(l) are the rule-based CCs. CUBIC, Vegas, and BBR's sending rates slowly approach the convergence direction. However, their flows do not converge until 30s under the current network scenario. The convergence of these rule-based CCs is largely affected by the network environment, e.g., the buffer size and the base-RTT of the link. Due to the adjustment of the ACK frequency, flows of Fillp are completely non-convergent. Anole also faces the problems of slow convergence of CUBIC and erroneous convergence points of Sage. However, whenever a new flow is added, the real-time sending rates of CUBIC and Sage will exceed the link capacity, and the decision on the sending rate will be made by inferring the optimal rate. In the process of inferring the optimal rate, Anole will estimate the current bandwidth share of other flow and reduce its own rate proportionally. After several iterations, Anole can achieve the convergence of multiple flows within 2s. In the subsequent operation process, since the rates of CUBIC and Sage still need to be evaluated, the throughput will still fluctuate. However, CUBIC does not disrupt the established convergence and Sage makes predictions based on the current rate, therefore, there will not be large fluctuations.

F. Coping With Bandwidth Changes

In this section, we conduct an experiment to evaluate the ability of CCs to quickly grab the spare bandwidth. The appearance of available bandwidth may come from modifications made by the network operators or the departure of the competing flows. We simulate this by changing the link capacity. The flow starts in a 48Mbps link, and the link capacity changes to 96Mbps at 15s. Fig. 10 shows the sending rate curves of the tested CCs. Due to the significant overlap in the rate curves of each CC, we divide the experimental results into Fig. 10(a) and Fig. 10(b).

In the face of spare bandwidth, PCC, Indigo, and Vegas show poor bandwidth adaption abilities. The rate of PCC and Indigo have hardly increased, maintaining a sending rate of 48Mbps though the link capacity has raised to 96Mbps. Vegas shows an increasing trend in the sending rate; however, the additive increase in cwnd severely limits its ability to preempt bandwidth. Vivace, Sage, Fillp, Remy, Libra, and Aurora quickly detect the appearance of available bandwidth, but it takes them a long time to occupy it. Fillp, Libra, and Aurora take about 2 seconds to occupy the available bandwidth, and the rate then fluctuates significantly.

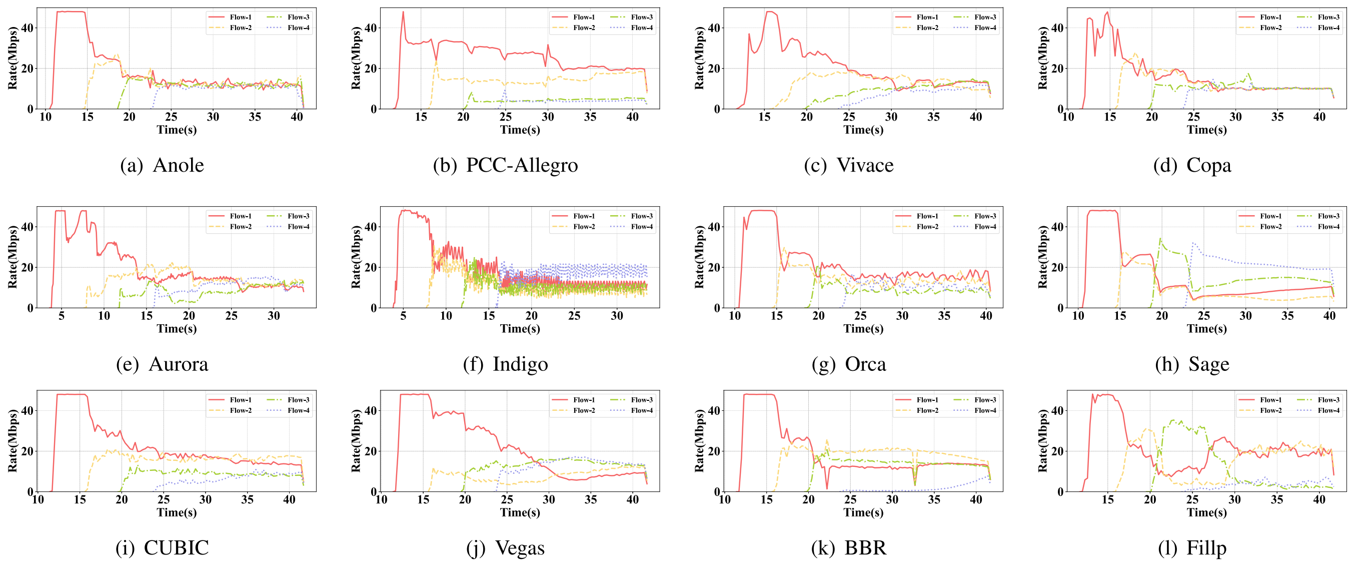


Fig. 9. Variation of the throughput as four flows gradually join the network at an interval of 4s. Anole significantly ameliorated the issue of suboptimal convergence in learning-based algorithms. The link capacity is 48Mbps.

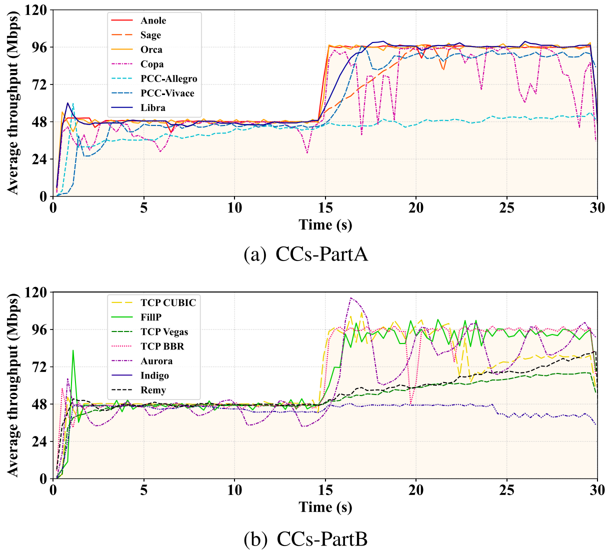


Fig. 10. The change of CCs' sending rates when link capacity changes. We divide the fourteen CCs into two graphs to display the curves more clearly. The experiment result shows that Anole can quickly grab the idle bandwidth.

Remy fails to fully utilize the available bandwidth until the flow ends. Vivace and Sage take 3s and 5s, respectively, to fully occupy the bandwidth. Anole, Orca, Copa, CUBIC, and BBR quickly detected the increase in available bandwidth and increased their sending rates to the link capacity within 0.5 seconds. However, during subsequent sending, Copa, BBR, and CUBIC all experienced varying degrees of rate oscillation. Anole and Orca outperform other CCs in this experiment, as they can maintain the best sending rate after quickly occupying the available bandwidth.

G. Time Allocation for Applied Rates

Fig. 11 shows the time fraction applied for the rule-based (CL), learning-based (RL), and Anole-inferred optimal (Opt)

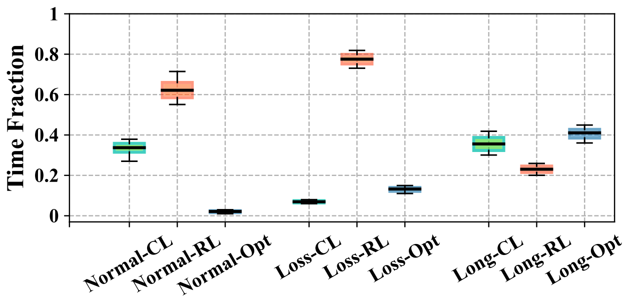


Fig. 11. The time fraction of rule-based, learning-based, and the referred optimal rate in a 30s flow. The flow in each network environment runs 100 times.

rates in the basic scenario (a 25ms link with no loss, denoted as Normal), loss scenario (a 25ms link with 0.1% loss, denoted as Loss) and long-link scenario (a 50ms link with no loss, denoted as Long). We run a single flow in each scenario for 100 times. Our statistical method for the time spent on each rate is: for example, if CUBIC is evaluated to have the maximum utility value in the normal scenario, CUBIC's rate will be applied in the probing stage (or the acceleration stage), then the time of the probing stage is recorded in Normal-CL. The statistics for RL and Opt are similar.

In the normal scenario, CUBIC's cwnd periodically exceeds the link's BDP, but under the buffer limitation, this cwnd overflow will not last for a long time. Meanwhile, Sage shows stable good performance. These indicate that there are limited situations where Anole needs to infer the optimal rate. Our log shows that the optimal rate inference only occurs when CUBIC's cwnd exceeds the BDP and Sage's prediction shows jitter. Fig. 11 shows the fraction of time applied for Normal-CL and Normal-RL is about 36% and 62% while the fraction of optimal rate is only about 2%.

In the loss scenario, CUBIC's rate is hardly applied in the probing and acceleration stage, which only runs for half an RTT

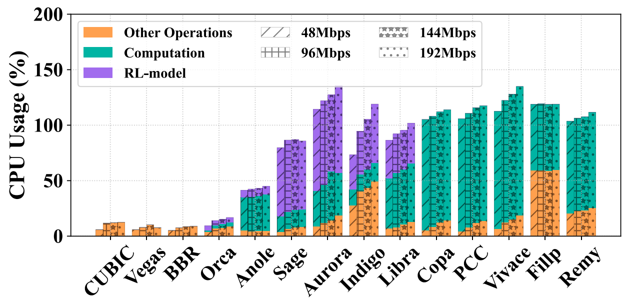


Fig. 12. The overhead of each CC under links with different capacities and the proportion contributed by each type of overhead.

in the evaluation stage in each round. Besides, CUBIC is often deprecated because of its bad performance, which further reduces the applied time to about 7%. For Sage, a 0.1% loss rate does not significantly impact its performance and only causes a few minor jitters in its prediction (usually resulting in lower predicted rates). Therefore, Anole applies Sage's rate in the majority of cases, approximately with a fraction of 80%, only inferring the optimal rate when Sage experiences prediction jitter.

In the long-link scenario, the fraction of applying the optimal rate increases to nearly 40%. This result is due to two aspects: 1) the slower rate adjustment of CUBIC. 2) the increased proportion of deviated rate prediction in Sage. CUBIC takes a longer time to recover from a cwnd reduction caused by packet loss, but the slower cwnd growth also means that CUBIC's rate curve is smoother and more stable. Therefore, overall, the proportion of time applying CUBIC's rate has increased, reaching about 35%. The severe oscillation in Sage's prediction reduces the probability that we can obtain an expected rate in the evaluation stage, leading to the applied fraction decrease to about 23%.

H. Overhead Analysis

Considering that the number of packets will affect the computational overhead in each CC, we record the overhead of each CC at links with different capacities: 48Mbps, 96Mbps, 144Mbps, and 192Mbps. We divide the overhead into three parts: computation overhead, RL-model overhead, and overhead caused by other operations. The computation overhead includes the overhead caused by data collection (e.g., counting the delay or loss information for RL-model prediction) and computation (e.g., target rate calculation and rate evaluation) in the user mode. The RL-model overhead refers to the overhead caused by invoking the RL-model and executing the prediction. The overhead caused by other operations includes the CC state maintenance, ACK generation and transmission, and the overhead caused by the interaction between user space and kernel space. Since the overhead of specific operations in the kernel-integrated CCs can not be counted separately, all the overheads of CUBUC, BBR, and Vegas are recorded as overhead caused by other operations.

Fig. 12 shows that as the link capacity increases, the overhead of all CCs shows a slight increase trend. The kernel-integrated CCs undoubtedly obtain the lowest CPU overhead. They always keep the CPU utilization below 20%, even in the 192Mbps link. For learning-based CCs, Orca shows the lowest CPU utilization, while Sage, Libra, Aurora, and Indigo have both high computational and RL-model prediction overhead. Copa, Fillp, PCC, Remy, and Vivace avoid the RL model inference overhead, but they all involve a large amount of computational overhead. Anole deploys Sage's RL model and needs to calculate the utility values in every control cycle while its overhead is still lower than most algorithms.

Analysis for overhead performance: 1) Computationally intensive CCs: According to the paper, Copa [29] updates the queueing delay and target sending rate at each ACK arrival (at every half RTT in the kernel implementation), involving a large amount of computational overhead. PCC [22] and Vivace [23] calculate the utility function in each RTT, and their utility-based rate adjustment strategy brings additional computing overhead. ***2) Learning-based CCs:*** Compared with other learning-based CCs, Orca deploys a smaller RL model with about 140k parameters. Based on Sage's source code, we found that its model parameter count is approximately 4.5 million, which is several hundred times that of Orca. The prediction intervals for these models are established as a configurable parameter, with a recommended value of 20 ms. This configuration suggests that multiple inferences are frequently performed within a single RTT. For online learning algorithms [9], [10], the necessity to continuously receive information transmitted by each acknowledgment (ACK) and to persistently update model parameters incurs considerable computational overhead, even for models of relatively small size. ***3) Anole:*** When rate inference is not executed, U is one of the U_{cl} and U_{rl} from the previous control cycle, eliminating the need for redundant calculations. Assuming the probing stage lasts for 3 RTTs, Anole only needs to compute U_{cl} and U_{rl} once within 5 RTTs. This approach saves 60% of computational resources compared to algorithms that require the utility value to be calculated every RTT. Additionally, Anole only needs to run the RL model once in the final RTT of each control cycle, resulting in at least an 80% reduction in overhead compared to other RL algorithms. Note that in Aonle, the overhead of the RL model is proportional to the number of flows. However, in the same link, the computational overhead increases only marginally with the number of flows. This is primarily because the computational load in Anole mainly arises from processing ACKs. Under constant link capacity, the number of ACKs does not exhibit significant variation.

VI. Related Work

Rule-based CCs: Rule-based CCs can be divided into loss-oriented and delay-oriented CCs. NewReno [28] and CUBIC [1], as the representatives of loss-oriented CCs, update their congestion windows in an Additive Increase Multiplicative Decrease (AIMD) manner. NewReno's addictive operation increases the cwnd with a single packet per RTT and halves the cwnd upon packet drop occurs, while CUBIC figures out the addictive step size according to a cubic function. Vegas [2], FastTCP [31], Westwood [32], and Copa [29] are delay-oriented CCs. The former three update their cwnd based on RTT measurements, which reduce the transmission rate when RTT growth is detected. Copa [29] proposes a target rate model and adjusts the cwnd in the direction of the obtained target rate. Note that Copa proposes a competitive mode to guarantee its competitiveness.

Learning-based CCs: Remy [11], Indigo [21], and Aurora [9] are the representatives of offline learning CCs. Based on a pre-specified objective for congestion control, Remy performs a table lookup to find the corresponding action. In its offline optimization phase, Remy pre-computes the lookup table by finding the mapping that maximizes the expected value of the objective function. Indigo observes the network states and adjusts the congestion window. Indigo uses a Long Short-Term Memory (LSTM) [33] recurrent neural network (RNN) to store the mapping from states to actions. Different from Indigo, Aurora is a rate-based CC based on DRL. Aurora employs a simple linear objective function and trains a DRL agent using Proximal Policy Optimization (PPO)[34]. PCC [22] and Vivace [23] are noteworthy online learning methods. They both learn in an online fashion using multiple micro-experiments, where the performance of each experiment is characterized by a utility function. The utility function of PCC comprises the throughput and loss rate, while Vivace further incorporates RTT gradient.

Hybrid CCs: In general, hybrid algorithms [3], [4], [16], [17], [18], [25] can be divided into two cases: (*i*) *Switching between alternate rates:* These algorithms [3], [4], [17], [25] run both the traditional and learning-based CCs separately and select one of the two rates based on the utility function (or at regular intervals). The first work on hybrid CC was proposed by Orca [3], which sets the sending rate to the learning-based CC after each monitoring period and lets the rule-based CC operate based on this rate. Subsequently, Libra [4] and MPLibra [25] further evaluate each rate using a utility function to determine the rate that performs better in the current environment. Spine [17] aims to reduce the overhead of learning-based CC by evaluating whether to invoke the machine learning CC at each monitoring period. (*ii*) *Deep integration by feature fusion:* To mitigate the uncertainty and lack of interpretability of learning-based CCs, such algorithms [16], [18] transform the rule-based CCs into equivalent black-box neural network (NN) models. Sage [18] collects the states and actions of multiple rule-based CC in different network scenarios and then uses them to train an offline DRL model. Loki [16] converts white-box rule-based CCs into black-box NNs to achieve feature-level fusion between the two types of CCs instead of decision-level fusion. These approaches demonstrate the diversity and creativity in hybrid CC but are largely limited by the performance of learning models.

DRL-model of Sage: Sage has collected a policy pool consisting of over 60 million data points across more than 1000 different environments by simulating 13 CC algorithms that have been implemented in the Linux kernel. Each data point is organized in the format of (state, action, reward), where the state is represented by a 69-dimensional input signal vector. This vector encapsulates the mean, maximum, and minimum values for three categories: delay-oriented, throughput-oriented, and loss-oriented input signals. The action is represented by the ratio of the congestion window (cwnd) at the current time step to the cwnd at the previous time step, emphasizing that Sage is more inclined to learn behaviors rather than merely recording specific values under varying settings. The reward is comprised of two components: one that focuses on optimizing the transmission quality for individual flows and another that rewards the degree of shared bandwidth to ensure fairness. After constructing the policy pool, Sage trains a deep neural network agent with two neural networks built on top of Encoders, GRUs(Gated Recurrent Units) and Residual blocks, and it outputs stochastic actions with a Gaussian mixture model (GMM). The training is finished using Critic-Regularized Regression (CRR) [26] on a general large-scale GPU cluster.

Broader Perspectives: Regarding network scenarios, the construction of satellite networks and low-altitude networks has dramatically enhanced the heterogeneity of the network. Due to the movement of satellites and drones, these networks exhibit highly dynamic characteristics. This requires CCs to be able to adapt to frequent link changes and achieve real-time link self-adaptation. Besides, the higher costs of satellite and low-altitude networks put forward a greater demand for fully utilizing the network. Regarding network technology, the promoting of the QUIC protocol provides possibilities for the research of cross-layer transmission algorithms. Applications can offer richer multi-dimensional features for optimizing the CCs in the transport layer. With these features, the transport layer can conversely meet the differentiated requirements of the application layer. We hope that researchers will invest more and progress in these future directions.

VII. Conclusion

In this paper, we present Anole, a three-stage hybrid CC framework to provide high throughput and low latency. Based on a refined utility function, Anole periodically determines the best rate from rule-based, learning-based, and previous decisions while deducing the optimal rate for the current scenario to achieve better performance. To address throughput degradation and high overhead, we will temporarily deprecate underperforming algorithms under the update of confidence score. Extension experiments on Pantheon have shown that Anole can achieve stable high performance and better convergence with lower overhead simultaneously compared with the state-of-the-art algorithms. We believe Anole hints at a new direction that infers a new sending rate based on the known rates and their performance.

Acknowledgments

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06, Shenzhen R&D Program under grant NO. KJZD20230923114059020, and in part by Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989.

References

[1] S. Ha, I. Rhee, and L. Xu, “Cubic: A new TCP-friendly high-speed TCP variant,” in *Proc. ACM SIGOPS*, 2008, pp. 64–74.
[2] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” in *Proc. SIGCOMM Comput. Commun. Rev.*, 1994, pp. 24–35.
[3] S. Abbasloo, C.-Y. Yen, and H. J. Chao, “Classic meets modern: A pragmatic learning-based congestion control for the Internet,” in *Proc. SIGCOMM*, 2020, pp. 632–647.

[4] Z. Du et al., “A unified congestion control framework for diverse application preferences and network conditions,” in *Proc. 17th CoNEXT*, 2021, pp. 282–296.

[5] N. Cardwell et al., “BBR: Congestion-based congestion control,” *Commun. ACM*, vol. 60, pp. 58–66, 2017.

[6] Y. Zaki et al., “Adaptive congestion control for unpredictable cellular networks,” in *Proc. ACM SIGCOMM*, 2015.

[7] S. Abbasloo, C.-Y. Yen, and H. J. Chao, “Wanna make your TCP scheme great for cellular networks? Let machines do it for you!,” *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 265–279, Jan. 2021.

[8] K. Winstein, A. Sivaraman, and H. Balakrishnan, “Stochastic forecasts achieve high throughput and low delay over cellular networks,” in *Proc. USENIX NSDI*, 2013, pp. 459–471.

[9] N. Jay et al., “A deep reinforcement learning perspective on internet congestion control,” in *Proc. 36th ICML*, 2019, pp. 3050–3059.

[10] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, “Pantheon: The training ground for internet congestion-control research,” in *Proc. USENIX ATC*, 2018, pp. 731–743.

[11] K. Winstein and H. Balakrishnan, “TCP ex machina: Computer-generated congestion control,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 123–134, 2013.

[12] S. Emara, B. Li, and Y. Chen, “Eagle: Refining congestion control by learning from the experts,” in *Proc. IEEE INFOCOM*, 2020, pp. 676–685.

[13] T. Meng et al., “PCC Proteus: Scavenger transport and beyond,” in *Proc. ACM SIGCOMM*, 2020, pp. 615–631.

[14] Z. Pan et al., “Marten: A built-in security DRL-based congestion control framework by polishing the expert,” in *Proc. IEEE INFOCOM*, 2023, pp. 1–10.

[15] X. Liao et al., “Astraea: Towards fair and efficient learning-based congestion control,” in *Proc. 19th Eurosys*, 2024, pp. 99–114.

[16] H. Zhang et al., “Loki: Improving long tail performance of learning-based real-time video adaptation by fusing rule-based models,” in *Proc. 27th MobiCom*, 2021, pp. 775–788.

[17] H. Tian et al., “Spine: An efficient DRL-based congestion control with ultra-low overhead,” in *Proc. 18th CoNEXT*, 2022, pp. 261–275.

[18] C.-Y. Yen, S. Abbasloo, and H. J. Chao, “Computers can learn from the heuristic designs and master internet congestion control,” in *Proc. ACM SIGCOMM*, 2023, pp. 255–274.

[19] H. Zhang et al., “OnRL: Improving mobile video telephony via online reinforcement learning,” in *Proc. 26th MobiCom*, 2020, pp. 1–14.

[20] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” 2018, *arXiv: 1802.09477*.

[21] F. Y. Yan et al., “Pantheon: The training ground for internet congestion-control research,” in *Proc. USENIX ATC*, 2018, pp. 731–743.

[22] M. Dong et al., “{PCC}: Re-architecting congestion control for consistent high performance,” in *Proc. 12th USENIX NSDI*, 2015, pp. 395–408.

[23] M. Dong et al., “PCC Vivace: Online-learning congestion control,” in *Proc. 15th USENIX NSDI*, 2018, pp. 343–356.

[24] P. Goyal et al., “Elasticity detection: A building block for internet congestion control,” in *Proc. ACM SIGCOMM*, 2022, pp. 158–176.

[25] H. Yu, J. Zheng, Z. Du, and G. Chen, “MPLibra: Complementing the benefits of classic and learning-based multipath congestion control,” in *Proc. IEEE 29th ICNP*, 2021, pp. 1–11.

[26] Z. Wang et al., “Critic regularized regression,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 7768–7778.

[27] V. Jacobson, “Congestion avoidance and control,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, pp. 314–329, 1988.

[28] S. Floyd, T. Henderson, and A. Gurtov, “The NewReno modification to TCP’s fast recovery algorithm,” Techn. Rep. rfc3782, 2004.

[29] V. Arun and H. Balakrishnan, “Copa: Practical delay-based congestion control for the internet,” in *Proc. 15th USENIX NSDI*, 2018, pp. 329–342.

[30] T. Li et al., “Tack: Improving wireless transport performance by taming acknowledgments,” in *Proc. ACM SIGCOMM*, 2020, pp. 15–30.

[31] C. Jin, D. X. Wei, and S. H. Low, “Fast TCP: Motivation, architecture, algorithms, performance,” in *Proc. IEEE INFOCOM*, vol. 4, Piscataway, NJ, USA: IEEE, 2004, pp. 2490–2501.

[32] C. Casetti et al., “TCP Westwood: End-to-end congestion control for wired/wireless networks,” *Wireless Netw.*, vol. 8, pp. 467–479, 2002.

[33] X. Shi et al., “Convolutional LSTM network: A machine learning approach for precipitation nowcasting,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, p. 1.

[34] J. Schulman et al., “Proximal policy optimization algorithms,” 2017, *arXiv: 1707.06347*.

Feixue Han received the B.S. degree from Beijing University, in 2018. She is currently working toward the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University. Her research interests include congestion control and traffic scheduling.

Yike Wang received the B.S. degree from the Northeastern University, in 2023. He is currently working toward the master’s degree with Tsinghua Shenzhen International Graduate School, Tsinghua University. His research interests include traffic scheduling and congestion control.

Yunbo Zhang received the B.S. degree from the University of Electronic Science and Technology of China, in 2024. He is currently working toward the master’s degree with Tsinghua Shenzhen International Graduate School, Tsinghua University. His research interests include congestion control and network measurement.

Qing Li (Senior Member, IEEE) received the B.S. degree from Dalian University of Technology, Dalian, China, and the Ph.D. degree from Tsinghua University, Beijing, China. Currently, he is an Associate Researcher with Peng Cheng Laboratory, Shenzhen, China. His research interests include reliable and scalable routing of the Internet, intelligent self-running network, and edge computing.

Dayi Zhao received the B.S. degree in computer science and technology from Xiangtan University, Hunan Province, China. He is currently engaged in network development with Pengcheng Laboratory. His research interests include network protocols, network transmission, network security, and aerospace networks.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China. Currently, he is a Full Professor with Tsinghua Shenzhen International Graduate School. His research interests include the future network architecture, the Internet QoS, and network function virtualization.