A Unified Congestion Control Framework for Diverse Application Preferences and Network Conditions

Zhuoxuan Du
State Key Laboratory for Novel Software Technology
Nanjing University

Jiaqi Zheng
State Key Laboratory for Novel Software Technology
Nanjing University

Hebin Yu
State Key Laboratory for Novel Software Technology
Nanjing University

Lingtao Kong
Ant Group

Guihai Chen
State Key Laboratory for Novel Software Technology
Nanjing University

ABSTRACT

With the increase of diversity in application needs and networks, existing congestion control algorithms (CCAs) do not accommodate this complicated reality. Previous classic CCAs are designed for a specific domain with fixed rules, failing to adapt to such diversities. Recently surged learning-based CCAs have great potential in adaptability and flexibility but are not practical due to unsatisfying performance on convergence, fairness, overhead and safety assurance. In this paper, we propose Libra, a unified congestion control framework, which empowers flexibility, adaptability, and practicality, by combining the wisdom of classic and reinforcement learning (RL)-based CCAs. Extensive evaluation of Libra's Linux kernel implementations on both live Internet and emulated networks shows performance improvement under dynamic networks (*e.g.*, 1.2× throughput than Orca on average). At the same time, Libra can flexibly satisfy different application needs, reduce the running overhead by at most 0.92× and perform good fairness and convergence properties, well-fitting our theoretical analysis.

CCS CONCEPTS

• **Networks** → **Transport protocols**; • **Computing methodologies** → *Reinforcement learning*;

KEYWORDS

TCP, Congestion Control, Deep Reinforcement Learning

ACM Reference Format:
Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, Guihai Chen. 2021. A Unified Congestion Control Framework for Diverse Application Preferences and Network Conditions. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3485983.3494840

CoNEXT '21, December 7–10, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3494840

1 INTRODUCTION

New waves of technique innovations drive a large number of emerging applications and make the network infrastructure significantly more diverse than before. For one thing, emerging applications require flexible preferences to guarantee high performance. Some applications such as replication of cloud storage and software downloading, are throughput-oriented, while others such as VR/AR and cloud gaming are delay-sensitive. For another, network communications are carried through different types of networks (*e.g.*, cellular networks, WIFI, or optical fiber) and different regions (*e.g.*, intra- or inter-continental). Hence, a modern CCA should adapt to such heterogeneity and keep pace with the times.

So far, at least 15 classic CCAs have been incrementally integrated into the Linux kernel [2], with a basic design principle of connecting possible congestion signals to fixed actions to be taken [15]. Classic CCAs are pragmatic as a result of their provable convergence and fairness, predictable behaviors, and negligible overhead. However, they are not designed to meet evolving application demands with a closed interface. Besides, these CCAs cannot perform well beyond the scope of their domain knowledge, *i.e.*, the throughput-oriented CCAs (*e.g.*, CUBIC), designed specifically for wired networks, usually perform poorly in cellular networks [1, 3, 34, 40].

Learning-based CCAs are expected to improve adaptability with their fully automatic rate calculation procedures. Meanwhile, the utility function is a potential interface that can be accordingly tuned to reflect different application needs. But state-of-the-art learning-based CCAs have not realized these ideal visions yet. First, their high performance is usually obtained under specific network environments [9, 20], showing limited adaptability. Their performance under wired and cellular scenarios also has room for further improvement (Sec. 2). Second, offline-trained CCAs [13, 20, 33] cannot flexibly tune the performance preferences mid-flow since their preferences are tightly coupled with trained models. Online-learning CCAs [11, 12] do not focus on application preferences. Proteus [25] can capture throughput demands but cannot tune latency preferences. Besides, learning-based CCAs experience unsatisfying performance in terms of convergence, fairness, overhead, and safety assurance [2, 28], which hinder large-scale deployments.

Based on the lessons we learned, we summarize three goals that a modern CCA should possess: (1) **Adaptability**: achieve high

performance under dynamic networks, including those unseen scenarios. (2) **Flexibility**: adjust performance preferences according to the application preferences. (3) **Practicality**: quickly converge to equilibrium, fairly share the bottleneck bandwidth, provide safety assurance and achieve acceptable overhead. To satisfy these goals simultaneously, we propose Libra, a unified framework for congestion control that combines the wisdom of classic and RL-based CCAs. Libra is motivated by the following observations: (1) Classic and RL-based CCAs have complementary advantages. RL-based CCAs have great potential in adaptability and flexibility without the need to be manually engineered. Classic CCAs can help them to realize these visions while mitigating their problems in practice. (2) Two heads are better than one. By evaluating and exploiting a better action of the two, Libra can become a performance frontier among all the CCAs.

The differences between Libra and Orca [2], an early attempt of the combined approach, lie in both the targets and designs. First, Libra includes the goal of Orca (*i.e.*, address convergence, overhead, and performance issues of pure learning-based approaches) as a subset of its objectives. Libra can realize these goals better while providing many new properties (*e.g.*, theoretical analysis, safety assurance, flexibility). Second, Orca enables the RL-based schemes to adjust the base cwnd of CUBIC directly. A key observation is that the DRL agent of Orca outputs unexpected rate decisions occasionally, which probably leads to severe performance degradation (Fig. 2(b)). Libra addresses this issue by leveraging a novel three-stage framework that efficiently evaluates and exploits the decisions of underlying CCAs. In Libra, the underlying classic CCA contributes to the properties of inter-protocol fairness and safety assurance in practicality. The RL-based CCA boosts the performance on adaptability. The design of our combined framework has low overhead, high performance, and provable inter-protocol fairness and convergence. In particular, our contributions are as follows.

First, we experimentally show the limitations of existing CCAs and describe the design of Libra in detail with several insights. Libra utilizes a three-stage control cycle, which (1) explores the network conditions, (2) evaluates the performance of both classic and learning-based CCAs using our utility-based framework, and (3) exploits the previous decision and determines the base sending rate of the next control cycle. Although the utility-based framework is similar to PCC [12] in high-level, our novelty lies in re-designing the algorithm with its rate control component and a comprehensive exploration of using our combination mechanism. With these improvements, Libra presents better adaptability, faster convergence speed, and lower overhead than PCC.

Second, considering the limited literature on revealing the impact of RL formulation (*i.e.*, the design of reward function, action space, and state space), we experimentally summarize key observations about it and accordingly optimize the RL-based component to improve Libra's performance. Then we discuss the details on the smooth integration of the classic CCAs and provide guidelines for tuning parameters.

Finally, we implement Libra in Linux kernel and evaluate it with state-of-the-art CCAs in both emulated networks and live Internet. Experimental results show that Libra can (1) consistently outperforms the state-of-the-art (*e.g.*, 1.2× throughput than Orca) in diverse networks, (2) have low sensitivity to the buffer size, stochastic loss and parameter settings, (3) flexibly tune the performance preferences according to the application demands, (4) reduce around 92% CPU utilization compared with other learning-based CCAs, (5) maintain Jain's fairness index over 98% in both inter- and intra-protocol fairness and quickly converge to a fair share.

2 PRELIMINARIES AND MOTIVATION

In this section, we experimentally show the limitations of existing CCAs. Experiments are done using Pantheon [38], a community evaluation platform, with default parameters for candidate CCAs. Here we use CUBIC as the underlying classic CCA for Libra.

Limitations on adaptability: To evaluate the adaptability of state-of-the-art CCAs, we deploy a combination of three wired network traces (24Mbps, 48Mbps, and 96Mbps bandwidth as Wired#1∼Wired #3) and three LTE cellular traces (stationary, walking, and driving scenarios as LTE#1∼LTE#3) with 30ms minimum RTT and 150KB buffer. We summarize two observations based on the results shown in Fig. 1. First, existing classic CCAs cannot adapt to various network conditions. For instance, CUBIC and BBR have bufferbloat and delay problems in cellular networks due to the throughput-oriented nature and inaccurate network models [3], respectively. Second, we observe that Orca and Proteus can reduce the average delay by up to 60% compared with CUBIC while achieving 8.4%∼13.5% lower link utilization than classic CCAs in LTE scenarios. We believe that the state-of-the-art learning-based CCAs still have room for improvement (*i.e.*, higher link utilization while remaining low delay). See Sec. 5.1 for more discussions.

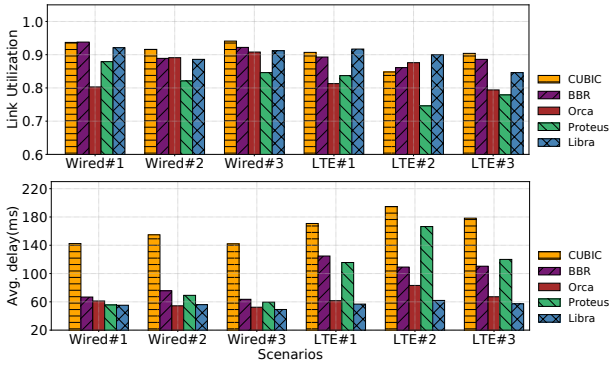
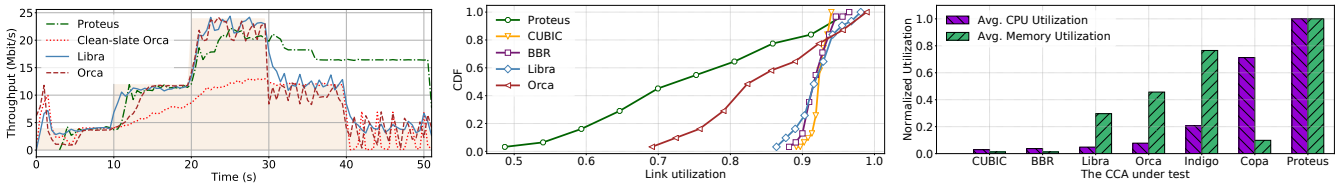


Figure 1: Adaptability Performance under wired / cellular networks

Limitations on flexibility: Classic CCAs are hard to dynamically adjust their behaviors in response to the varying application preferences. On the one hand, the hardwired mapping between predefined events and actions makes the classic CCAs hard to be tuned. For instance, it is not feasible to maintain a low queuing delay for CUBIC without the involvement of AQM schemes (*e.g.*, CoDel [27]) which requires changes in the network devices and incurs extra costs. On the other hand, the interface of adjusting the application preferences is closed for classic CCAs. Although learning-based approaches enable this interface to some extent, they only focus on a specific demand (throughput or delay [3, 25]) and cannot adjust multiple preferences simultaneously.



(a) Throughput comparisons over the step-scenario

(b) CDF of link utilization over cellular networks

(c) Normalized overhead comparisons

Figure 2: Practical issues for existing CCAs.

Limitations on practicality: Classic CCAs always obtain good practicality. In contrast, no learning-based CCA has been transferred to real-world production systems due to their practical problems. To shed light on these problems, we first consider a scenario whose available capacity changes every 10 seconds (step-scenario) with an 80ms minimum RTT and 1BDP buffer. From Fig. 2(a), we can observe that these CCAs cannot converge to the link capacity perfectly (*i.e.*, 30~50s for Proteus and 40~50s for Orca). We attribute it to (1) the inappropriate DRL decisions of Orca that cannot fully utilize the available bandwidth when the link capacity (5Mbps) is beyond the scope of its training environments (6~192Mbps) and (2) the lack of agile mechanisms to quickly identify the change of the equilibrium points for Proteus [2]. Second, we measure the CDF of link utilization over 100 repeated experiments in an TMobile LTE cellular network (0~40Mbps, 30ms RTT, 150KB buffer [3]). As shown in Fig. 2(b), Orca and Proteus exhibit a highly variable performance in our repeated experiments, failing to provide safety assurance. We think that it is caused by the uncertainty of the decision-making procedure of learning-based components. Third, the computational overhead is a common concern since learning-based CCAs usually involve complex computation to make decisions. Here we briefly show the measurement of the normalized average CPU and memory utilization when sending traffic on the same TMobile LTE networks as Fig. 2(b) for 60 seconds. Fig. 2(c) illustrates that existing pure learning-based CCAs have high CPU and memory overheads (*e.g.*, 88.7% and 10.1% for Proteus, 18.3% and 7.2% for Indigo). Finally, pure learning-based CCAs also perform poorly in both inter- and intra-protocol fairness (Fig. 13 and Fig. 14). More details and explanations will be discussed in Sec. 5.3.

Summary: Classic CCAs have good practicality, but are hard to achieve adaptability and flexibility. Learning-based CCAs can essentially achieve good adaptability, but confront a series of practical problems. The complementary advantages of classic and learning-based CCAs motivate us to propose a combined framework. Note that we use RL because it is sequential and far-sighted, which fits the sequential decision-making process of CC problems well [1, 24].

3 LIBRA OVERVIEW

On a high level, Libra adopts a utility-based framework to quantify the performance of different sending rates, which calculates a utility value for each candidate and chooses the best one eventually. Generally, this utility function includes three critical variables — throughput, latency, and loss [22] — to capture network conditions, with an objective of awarding the behavior that leads to lower latency, less loss, and higher throughput [12]. Note that the utility function plays an important role in maintaining Libra's convergence and intra-protocol fairness which will be soon discussed in Sec. 4.1.

Fig. 3 shows a basic control cycle of Libra and it includes an exploration stage, an evaluation stage, and an exploitation stage, where the lines colored red, violet and green represents the determined sending rate of classic CCA, learning-based CCA, and the winner (initial base sending rate) in the last control cycle, respectively. The solid line indicates the changes in sending rate applied by Libra and the dashed violet line is suggested by the learning-based CCA.

Basically, at the exploration stage, Libra initially sets the base sending rate as the winner in the last cycle and follows the classic CCA to adjust the sending rate. At the same time, the network feedback information is input into the learning-based CCA and it works as a backup. At the evaluation stage, Libra tries the sending rate of the classic CCA and the learning-based CCA one by one for a while to perform the evaluation. Finally, in the exploitation stage, Libra uses the initial base sending rate determined in the last control cycle and waits for the feedback information coming from the candidate rates in the evaluation stage. By gathering and transforming the statistics into utility values, Libra chooses the best sending rate and applies it as a new base sending rate in the next control cycle. However, for some special cases, Libra senders

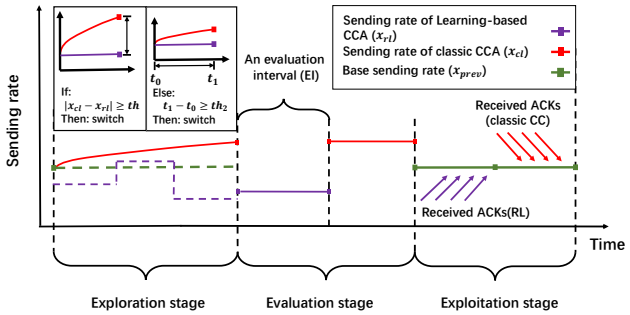


Figure 3: Time-diagram of Libra.

may not receive any ACKs during a decision-making interval as the network conditions (*i.e.*, available bandwidth, or the number of competing flows) may change midway, or the ACKs might be occasionally delayed [2]. Libra handles these cases in different ways. Specifically, if there is no ACK received during the exploration stage, Libra will skip the corresponding actions and maintain the same rate decision x_{rl} for the learning-based CCA. While in the case of

no ACK received in other stages, Libra cannot accurately evaluate the rate decisions and thus repeatedly use the current base sending rate x_{prev} in the next control cycle.

4 LIBRA DESIGN

In this section, we first describe the design of our combined framework in detail. In what follows, we experimentally summarize key observations about the formulation of RL-based CCA and discuss the details on the integration of the classic CCAs into Libra.

4.1 Combined framework

Libra's combined framework includes three stages in its control cycle and adopts a well-designed utility function to evaluate the rates from underlying CCAs.

Exploration stage: In this stage, starting with a base sending rate x_{prev} from the last control cycle, a classic CCA updates the sending rate of Libra in a per-ACK manner. At the same time, the network feedback information is input into a learning-based CCA and it works as a backup in a per-MI manner. Our purpose is to explore the network conditions from classic CCAs and learning-based CCAs and complement their own advantages.

When should Libra exit the exploration stage? As shown in Fig. 3, two cases can trigger Libra to exit the exploration stage — the divergence between two candidate decisions is large enough (*i.e.*, $|x_{rl} - x_{cl}| \geq th$) or the maximum duration of the exploration stage is reached. Both of the cases indicate that it is the moment to evaluate the decisions from underlying CCAs again, to timely respond to the changing network conditions.

Evaluation stage: Libra spends two evaluation intervals (EIs) to resolve the divergences at the evaluation stage. As shown in the middle of Fig. 3, one EI applies the rate x_{cl} from the classic CCA and the other one applies the rate x_{rl} from the learning-based CCA. At the same time, Libra gathers the network statistics and calculates a utility value $u(x_{prev})$ corresponding to the behavior in the exploration stage, which is used to compare with $u(x_{cl})$ and $u(x_{rl})$ at the end of the exploitation stage. Different from the first stage, Libra no longer calculates the sending rate from both the classic CCA and the learning-based CCA. This can greatly decrease the computation overhead by reducing expensive learning-based calculations, and thus benefit the practicality (Sec. 5.3).

Can Libra mitigate the side effect of different evaluation orders? Note that the evaluation order of the candidate rates should be carefully decided — an inappropriate evaluation order can result in a side effect on the final decision. As shown in Fig. 4(a), two candidate rates x_{cl} and x_{rl} are both beyond the available capacity and without loss of generality, we assume that $x_{cl} > x_{rl}$. First trying x_{cl} and then x_{rl} will lead to $u(x_{cl}) > u(x_{rl})$. Actually, the lower rate x_{rl} is better, whose utility value becomes smaller due to the produced side effect (increased delay and loss with the accumulated queue) when applying the rate x_{cl} first. On a reverse order — trying x_{rl} first and then x_{cl}, we can mitigate the side effect and obtain the right decision. Similar observations can also be found in Fig. 4(c) and Fig. 4(d). Hence, trying the lower rate first is always a correct evaluation order. The guideline behind this "lower rate first" principle is to minimize the self-inflicted side effect from testing the candidate rates, avoiding wrong decisions. Besides the evaluation orders, the competing background flows and the link capacity variations can also affect the performance statistics. Libra can react to this kind of dynamics by exploring the network condition in the first stage of the next control cycle and accordingly update the candidate rate decisions.

Exploitation stage: At the exploitation stage, Libra exploits the sending rate x_{prev} determined in the last control cycle. Meanwhile, the feedback (*i.e.*, ACK) of applying candidate rates during the evaluation stage return and thus Libra can calculate their utility values $u(x_{rl})$ and $u(x_{cl})$ at this point. This is why we neatly postpone the exploitation of x_{prev} and let it be after the evaluation stage, as shown in Fig. 3. At the end of the exploitation stage, Libra applies the sending rate with the highest utility value among the rates x_{prev}, x_{cl} and x_{rl} and enters into the next control cycle. In this way, Libra periodically updates its base sending rate, which ensures that the performance is no worse than that of both classic and learning-based CCAs, mitigating the unexpected behaviors of learning-based CCAs.

Utility function: Libra leverages the utility function to evaluate the decisions from its underlying CCAs. By default, Libra's utility function used in the evaluation stage is:

$$u(x_i) = \alpha \cdot x_i^t - \beta \cdot x_i \cdot max\left\{0, \frac{d(RTT_i)}{dt}\right\} - \gamma \cdot x_i \cdot L \qquad (1)$$

where $0 < t < 1$, the preference parameters $\alpha, \beta, \gamma > 0$, x_i is the sending rate of sender i and L is the observed loss rate. We argue that with reasonable design, Libra can converge to a unique equilibrium point [10] — each sender maintains a stable and fair sending rate with maximum utility value.

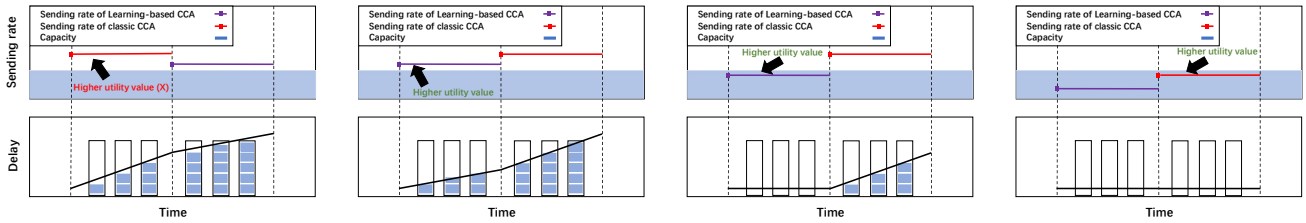
Theorem 4.1. **Convergence and Fairness:** *Under droptail queue, n Libra senders can finally achieve unique Nash equilibrium — a stable fair share* $(\overline{x}_1, \cdots, \overline{x}_i, \cdots, \overline{x}_n)$ *of the bottleneck bandwidth. Formally, for any sender i and any non-negative sending rate* x^**, we can derive that*

$$u_i(\overline{x}_1, \cdots, \overline{x}_i, \cdots, \overline{x}_n) > u_i(\overline{x}_1, \cdots, x^*, \cdots, \overline{x}_n)$$

where $\overline{x}_1 = \overline{x}_2 = \cdots = \overline{x}_n$ *and* $\sum_{i=1}^{n} \overline{x}_i \geq C$.

The proof of Theorem 4.1 roughly consists of two parts. Specifically, we justify the existence and uniqueness of Nash equilibrium in the first part and prove that Alg. 1 enables it to converge to this equilibrium in the second part. Detailed proof can be found in Appendix A. Note that any utility function that enables the existence of the unique equilibrium point can be used in Libra's evaluation stage without harming the convergence and fairness. A large number of utility functions, such as those used in PCC series [11, 12, 25] and Owl [29], all belong to this scope.

Algorithm: At each control cycle, Libra does the following. First, Libra starts with a base sending rate x_{prev}, which is obtained from the last control cycle (line 4). In the first stage, Libra calculates the sending rate x_{cl} and x_{rl} from the learning-based CCA and the classic CCA, respectively (lines 5-9), and applies x_{cl} to the current sending rate. Before Libra applies the current sending rate, it examines the difference between x_{cl} and x_{rl} and decides whether entering into the evaluation stage in advance or not (lines 10-11). During the evaluation stage, Libra spends two EIs trying the candidate sending rate x_{cl} and x_{rl} one by one, one EI for one candidate rate and the



(a) Case 1: the side effect is major (b) Case 1: the side effect is minor (c) Case 2: no side effect (d) Case 3: no side effect

Figure 4: The side effect of different evaluation orders on the utility values.

Algorithm 1: Libra

Input: the decided sending rate of the previous control cycle x_{prev}; classic CCA $CC_{classic}$; current measured state vector S_t; the weight parameters of reward function w_1, w_2 and w_3; the history length h, the length of exploration stage k, the difference threshold of the decisions th_1.

1 **foreach** *control cycle t* **do**
2 //Exploration stage
3 **for** *the time in the first k estimated RTTs* **do**
4 Initially set the sending rate x_t as x_{prev};
5 **foreach** *RL's decision-making interval* **do**
6 $x_{rl} = DRL\_based\ CCA(S_t, w_1, w_2, w_3, h)$;
7 **foreach** *Classic CC's decision-making interval* **do**
8 $x_{cl} = Classic\_CCA(CC_{classic}, S_t)$;
9 Update sending rate with x_{cl};
10 **if** $|x_{cl} - x_{rl}| \geq th_1$ **then**
11 Break, turn to the evaluation stage.
12 //Evaluation stage
13 Try a smaller rate between x_{rl} and x_{cl} first for one EI.
14 Try the remaining one then for another EI.
15 Collect the performance statistics, calculate utility value $u(x_{prev})$.
16 //Exploitation stage
17 **for** *the time in the next k estimated RTTs* **do**
18 Send traffic with rate x_{prev}.
19 Collect the performance statistics corresponding to x_{cl} and x_{rl}, respectively.
20 //At the end of this control cycle t.
21 Calculate the utility value $u(x_{cl})$, $u(x_{rl})$.
22 $x_{prev} = \arg\max_{x_{prev}, x_{rl}, x_{cl}} \{u(x_{prev}), u(x_{rl}), u(x_{cl})\}$;
23 $t = t + 1$;

lower rate first (lines 13-14). At the same time, Libra receives the network feedback and calculates the utility value $u(x_{prev})$ (line 15). Finally, Libra uses the sending rate x_{prev} and calculates two utility values $u(x_{rl})$ and $u(x_{cl})$ evaluated before, choosing the rate that leads to the highest utility as the base sending rate in the next control cycle (lines 17-23).

4.2 RL-based CCA

We reveal the impact of RL formulation and re-design each component in RL-based CCA in order to improve Libra's overall performance. We use a default setting of 100Mbps, 100ms RTT, and 1 BDP buffer for the following experiments.

State space: State space can represent the outside environment and we can analyze the benefits attained from previous actions. In this part, we aim to design a better state space for the RL-based CCA. We list the state choices attained from previous popular learning-based CCAs in Tab. 1.

Table 1: State candidates of learning-based CCAs

Index	State Candidates
(i)	EWMA of the time gap between two sequential ACKs [21, 33]
(ii)	EWMA of the timestamp difference between two sequential packets [2, 21, 33]
(iii)	Ratio between the most recent and minimum RTT [20, 21, 33]
(iv)	Current sending rate [2, 21, 25, 37]
(v)	Ratio between the packets sent and acknowledged [20]
(vi)	Current RTT and the minimum RTT [2, 37]
(vii)	Average loss rate of packets [2, 25]
(viii)	Derivative of latency with respect to time [20, 25, 37]
(ix)	Average delivery rate [2, 37]

Based on this, we evaluate the performance of different state spaces used in previous learning-based CCAs as shown in Fig. 5. We can observe that the state spaces of DRL-CC and PCC obtain a higher reward among all these CCAs. Hence, we further explore the state space used in PCC and DRL-CC and produce a baseline — state (iv), (vi), (vii), (viii), and (ix) —- by uniting the states of them. Then we search possible state combinations by using the simulated annealing algorithm [19] to improve the baseline design. Tab. 2 shows part of our results that are listed in descending order by rewards — the performance comparisons in terms of normalized rewards, throughput, latency, and loss rate respectively when adding to or removing from states in the baseline. Note that removing the state (vi) from our baseline produces the best reward among all the state combinations that we have searched. We find that a new state space combination (iv), (vii), (viii), and (ix) can better represent the network conditions and achieve high performance among all the candidates.

Table 2: Performance comparisons when adding or removing states in the Baseline that includes state (iv), (vi), (vii), (viii) and (ix)

State	Reward	Throughput	Latency	Loss
Baseline	0%	0%	0%	0%
-(vi)	+5.1%	-0.3%	-1.1%	-4.3%
+(i)(ii)	+3.7%	-0.7%	-1.8%	-2.6%
+(i)(ii)(iii)	+2.9%	-0.3%	-0.4%	-2.8%
+(ii)(iii)(v)-(iv)	+1.1%	+2.6%	+3.0%	-1.5%
+(iii)	-9.5%	+0.3%	+2.3%	+7.5%
+(ii)	-9.8%	+0.4%	+2.4%	+7.9%
+(i)	-12.4%	+0.3%	+3.1%	+9.7%
-(ix)	-14.4%	-0.2%	+2.3%	+11.8%

At the same time, when applying this new state space combination to Libra, we can observe clearly from Fig. 5 that it performs the best among all the previous CCAs. We also normalize these statistics in our state space to achieve better generalization. Besides, considering that the sender can only infer the network statistics from the observed features [2], we construct the state vector with several previous states, instead of using just the most recent one, allowing the sender to capture the dependency in the sequential data and detect the changes of network conditions [20]. Therefore, we combine h normalized feature vector f and formulate the state vector as $S = \langle f_{t-h+1}, f_{t-h+2}, \cdots, f_t \rangle$.

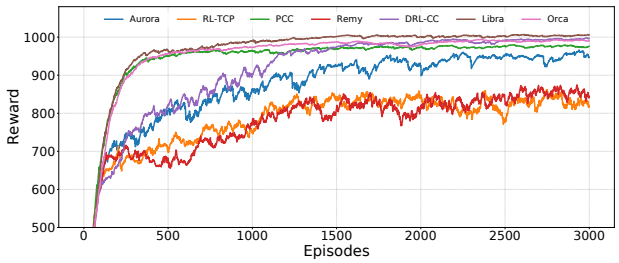


Figure 5: The reward comparisons of different CCAs' state combinations.

Action space: The action set of RL-based CCAs follows either AIAD mode (*e.g.*, RL-TCP and DRL-CC) or MIMD mode (*e.g.*, Aurora and Orca[1]). Specifically,

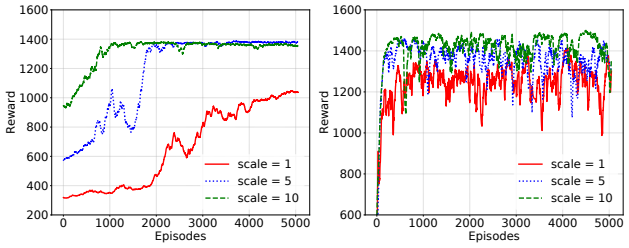
$$x_{t+1} = x_t + a_t \quad \text{for AIAD}$$

$$\text{or} \quad x_{t+1} = \begin{cases} x_t \cdot (1 + \delta a_t) & \text{if } a_t \geq 0 \\ x_t \,/\, (1 - \delta a_t) & \text{otherwise} \end{cases} \quad \text{for MIMD (Aurora)}$$

$$\text{or} \quad x_{t+1} = x_t \cdot 2^{a_t} \quad \text{for MIMD (Orca)}$$

where a_t indicates the action generated by the DRL agent, δ is a scaling factor and we set δ to 0.025. We evaluate the performance of AIAD and MIMD mode with three different scale factors: $scale = 1$ ($a_t \in [-1, 1]$), $scale = 5$ ($a_t \in [-5, 5]$) and $scale = 10$ ($a_t \in [-10, 10]$). As shown in Fig. 6, we can observe that all action sets can achieve similar rewards (around 1400) except the red line with a small scale factor $scale = 1$. Specifically, the action set with

[1]The action space of Orca is $a_t \in [-2, 2]$ which leads to a MIMD adjustment on the sending rate ($[\frac{1}{4}, 4]$)



(a) AIAD mode (b) MIMD mode

Figure 6: The reward comparisons of different action space designs.

AIAD mode consumes more episodes to learn the policy at the beginning. On the contrary, the action set with MIMD mode learns the policy faster and can quickly converge to a stable state. Despite a slight fluctuation of rewards shown in Fig. 6(b), we can mitigate it with our combined framework and finally choose the MIMD action space in our RL-based CCA.

Reward function: Here we report two key observations about the design of reward functions. The first observation is that the loss rate is a key variable in the reward function, though DRL-CC and Remy cannot include it. Intuitively, the loss rate is tightly related to the increasing latency especially under drop-tail queues and one may argue that just including latency is enough. However, for the RL-based congestion control shown in Tab. 3, the reward function without loss rate leads to worse performance (higher latency and loss). Through deep analysis, we conclude two reasons: (i) RL-based congestion control adjusts sending rate per MI to collect network feedback and make decisions, rather than traditional per ACK adjustment. Hence, the average RTT used in the reward function cannot exactly reflect what happened inside an MI. (ii) The utility function without loss rate cannot work when the queue is full. At this point, the utility value keeps unchanged even if the RL agent continues increasing the sending rate.

Table 3: Comparisons with and without the loss rate.

Setting	Throughput	Latency	Loss rate
with loss rate	97.2Mbps	115ms	0.72%
w/o loss rate	98.9Mbps	197ms	37.5%

Next, we provide two options when designing the reward function — one takes the current reward value r and the other uses the difference Δr between two consecutive reward values. Aurora and Orca appreciate the former design, while RL-TCP prefers the latter. The reason to take the difference as a reward is that it can better represent the improvement caused by the corresponding actions. For example, an action going on to increase its sending rate obtains a smaller positive reward value r but a negative Δr in the case of a near fully-utilized link. The essential reason is that the throughput in reward functions may keep unchanged but latency and loss rate become larger. At this point, when using r as the final reward, the

RL agent wrongly encourages this action, leading to poor performance in terms of latency and loss rate. Similarly, when new flows arrive, the previous flows react slowly — they usually continue to maintain a sending rate close to the bandwidth due to the still positive r, resulting in extreme unfairness [2]. The performance comparisons using r and Δr are shown in Tab. 4. We observe that using Δr can improve the performance on average latency and loss while maintaining a similar high throughput. However, fairness is a limitation for RL-based CCAs so far and it is hard to significantly improve fairness by RL-based CCA itself — this inspires us to integrate classic CCAs and use a combined approach to improve fairness.

Table 4: Comparisons using r and Δr.

Setting	Throughput	Latency	Loss rate	Fairness
r	99.4Mbps	173ms	14.7%	0.741
Δr	98.1Mbps	121ms	0.91%	0.780

Based on the discussions above, we design our RL-based CCA as shown in Alg. 2. The RL-based CCA takes the collected performance statistics, the weights of the reward function, and the history length as the input. It periodically updates the state vector S_t and calculates the reward corresponding to the previous action (lines 1-3). By utilizing the Proximal Policy Optimization (PPO) algorithm [30], it learns to output a final rate x_{rl} (line 4). At the same time, we update the maximum throughput x_{max} and the minimum delay d_{min} for normalizing the reward (line 6).

Algorithm 2: DRL_based CCA

Input: the current feature f_t; the weight parameters w_1, w_2 and w_3; the history length h.

Output: RL decision on sending rate x_{rl}

1 Update the state vector $S_t = \langle f_{t-h}, ..., f_{t-1} \rangle$ with current state f_t;

2 Obtain the throughput x_t, delay d_t and loss rate L_t from S_t $r_t = w_1 x_t / x_{max} - w_2 d_t / d_{min} - w_3 L_t$;

3 $R_t = r_t - r_{t-1}$;

4 $x_{rl} = PPO(R(t), S_t)$;

5 //PPO is the RL algorithm used in our RL-based CCA.

6 Update x_{max} and d_{min}

4.3 Classic CCA

As a combined framework, Libra takes the classic CCA as a subroutine (Alg. 1). The challenge is how to integrate classic CCAs without hurting their own advantages, which involves setting the duration and the threshold to exit Libra's exploration stage while unifying the window-based or rate-based schemes when combining different CCAs. First, most classic CCAs (*e.g.*, CUBIC) calculate an increase or a decrease based on the current sending rate, rather than from the scratch. These CCAs can be easily integrated into Libra with almost no modifications. We set the exploration stage to one RTT for them, which enables Libra to explore a wide range of available bandwidth while quickly reacting to the network dynamics. A special case is BBR, which probes the available bandwidth by adjusting its sending rate to 1.25×, 0.75×, and 1× for six times, with a total of 8 RTTs. We inherit the first three RTTs from BBR's control loop into Libra's exploration stage since they embody the main function of the bandwidth probing procedure. Second, the threshold (th_1) is set to 0.3× base sending rate to cover the bandwidth probing phase of BBR (±0.25× rate) and react to severe congestion observed by underlying classic CCAs.

5 EVALUATION

Implementation: The RL-based CCA of Libra is implemented in user-space, while the classic CCA of Libra is implemented in Linux kernel and we choose CUBIC (C-Libra) and BBR (B-Libra) due to their popularity. We adopt two fully-connected hidden layers with 512 neurons on each layer for both the critic and actor networks. They are trained using the Proximal Policy Optimization (PPO) algorithm on top of Tensorflow (1.14.0) [18]. The training environment used for Libra emulates a wide variety of networks with four parameters: link capacity (10 ∼ 200Mbps), minimum RTT (10 ∼ 200ms), buffer size (10KB ∼ 5MB), and stochastic loss rate (0 ∼ 10%). We randomly reset these network characteristics and start a new flow when entering a new training episode.

Setup: We evaluate Libra in both emulated networks and live Internet. For the emulation, we use Mahimahi emulator [26] with Stanford Pantheon [38]. For the real-world evaluation, we create several Amazon EC2 instances located in Tokyo, Hong Kong, Mumbai, Eastern US, and Southern US to transmit data to each other. The parameters of the utility function [12] are $t = 0.9, \alpha = 1$, $\beta = 900$, and $\gamma = 11.35$, and the weights of the reward function are $w_1 = 1, w_2 = 0.5, w_3 = 10$ in the DRL-based CCA. Furthermore, the duration of each stage for CUBIC is 1 RTT, and that for BBR is 3 RTT, 1 RTT, and 3 RTT, respectively. Detailed analysis about the choice of parameters is presented in Sec. 7. We compare various state-of-the-art CCAs with Libra including CUBIC [16], Sprout [6], Copa [5], BBR [8], Proteus&Vivace [25], Remy [31], Aurora [20] and Orca [2] and use their default settings in our experiments. We add another two benchmark CCAs, Clean-Slate Libra (CL-Libra) and Modified RL (Mod. RL), to emphasize the importance of combination and show the shortages of solely applying Eq. 1, respectively. Unless stated otherwise, the results are calculated based on the average of 5 runs, each lasting for 60 seconds.

5.1 Performance on Adaptability

We report the performance of benchmark CCAs under different traces, buffer sizes and stochastic loss rates.

Impact of different traces: We use four wired traces (12Mbps, 24Mbps, 48Mbps, and 96Mbps bandwidth), and four LTE traces collected by Pantheon and DeepCC [3] to emulate wired and cellular networks, respectively. The performance for each CCA is aggregated over wired and cellular networks and the average results are shown in Fig. 7. We can observe that both C-Libra and B-Libra are located close to the top right region of the figure, achieving the Pareto dominate results. Specifically, Libra shows a significant improvement compared with its underlying CCAs, pure learning-based and Orca. First, C-Libra achieves a similar throughput (0.97/0.95×) but

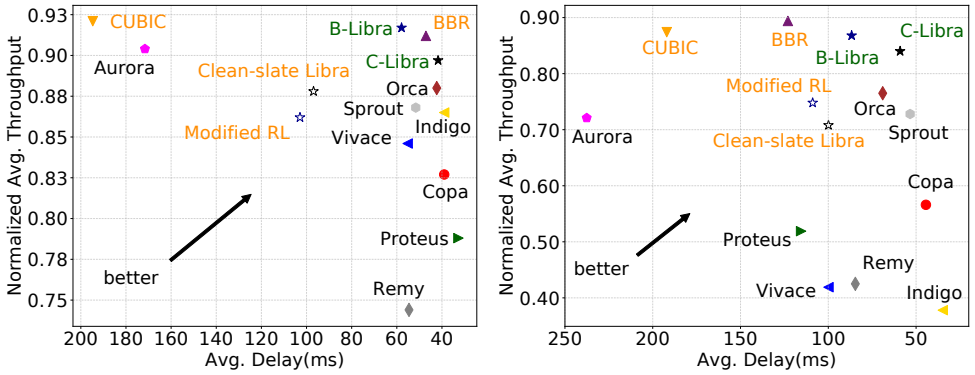


Figure 7: Average throughput and delay on (a) four wired traces and (b) four cellular traces.

far less delay (4.6/3.3× lower) than CUBIC in wired and cellular scenarios. B-Libra reduces the delay by 30% on average in cellular networks, while obtaining similar high performance in wired networks than BBR. Both of them beat the Clean-Slate version and Modified RL in our experiments. Second, we observe that existing learning-based CCAs (*e.g.*, Vivace, Proteus, Remy, Indigo, Aurora) excel in parts of four cellular traces but give a poor performance on others. Therefore, they do not achieve high performance as shown in Fig. 7. Besides, the overall throughput of Orca is still below what Libra obtains in emulated scenarios.

To present the adaptability of Libra in response to the link capacity variations in LTE scenarios, we evaluate Libra with a cellular trace involving the user movement [3]. As shown in Fig. 8, we can observe that Libra can perfectly adapt to the dynamic network capacity. Other benchmark CCAs may either over- or under-utilize the link capacity (*i.e.*, CUBIC: 20~30s, Orca: 20~25s, BBR: 10~15s) or even cannot follow the capacity (*i.e.*, Proteus).

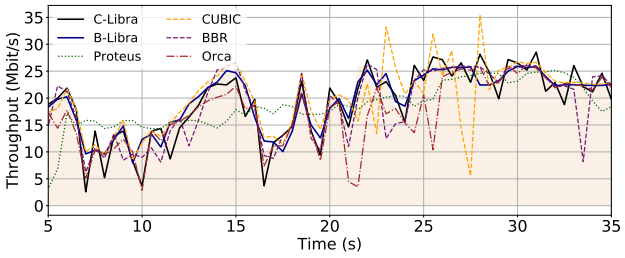


Figure 8: Performance when following the changing link capacity in LTE networks.

Remark 1: The results shown in Fig. 7 clearly illustrate the advantages of combining the wisdom of classic CCAs with RL-based CCAs. Specifically, the good decisions from Libra's optimized RL component help Libra outperform its underlying classic CCAs by controlling the delay better, while those from underlying classic CCAs in turn improve Libra's adaptability over pure learning-based CCAs by handling unseen scenarios better. Besides, Libra realizes the objective of adaptability better than Orca mainly due to the evaluation mechanism in the combined framework, which successfully avoids Orca's performance degradation caused by inappropriate decisions from the DRL component.

Impact of different buffer sizes: Existing CCAs often face the dilemma between high link utilization and low delay: the deep-buffered switches can achieve high link utilization, while this in turn leads to higher delay. We vary the buffer size from 10KB to 1MB in a 60Mbps emulated network with 100ms RTT. As shown in Fig. 9, when the buffer buildups, the link utilization, and the average delay for CUBIC increase. BBR also experiences a slightly higher delay when the buffer is deep. As is expected, Libra shows a significant improvement and has lower sensitivity to the buffer size than CUBIC and BBR. Similar to Proteus, Libra can obtain over 80% link utilization with only a 30KB buffer size.

Remark 2: CUBIC periodically fills and drains the buffer, leading to a higher queuing delay when buffer size increases. Libra breaks this dilemma via the help of the RL component and combined framework. On the one hand, Libra's underlying RL-based CCA can take precautions before filling the buffer. On the other hand, its combined framework prefers the decisions with lower delay, which prevents Libra from adding the queue length even in the presence of deep-buffered switches, showing very low sensitivity to the buffer size.

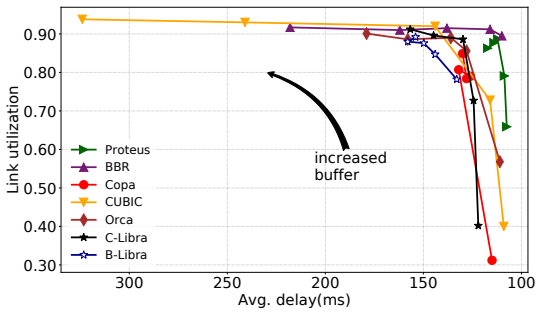


Figure 9: Impact of buffer size on performance.

Impact of stochastic packet loss: We further demonstrate that the stochastic packet loss has a minor impact on the average throughput for Libra. We vary the stochastic loss rate from 0% to 10% [12].

As shown in Fig. 10, B-Libra can maintain high link utilization (81.9%) when the stochastic loss rate is set to 10%. C-Libra also outperforms both CUBIC and Orca in presence of the stochastic packet loss.

Remark 3: B-Libra achieves high resilience to stochastic packet loss since its underlying CCAs do not treat a single loss event as a congestion indicator. C-Libra can immediately correct the erroneous reduction caused by the stochastic packet loss with the help of x_{rl} and x_{prev}, which always results in higher utility value than that using x_{cl}. It is also a more effective mechanism to recover from the stochastic packet loss than Orca.

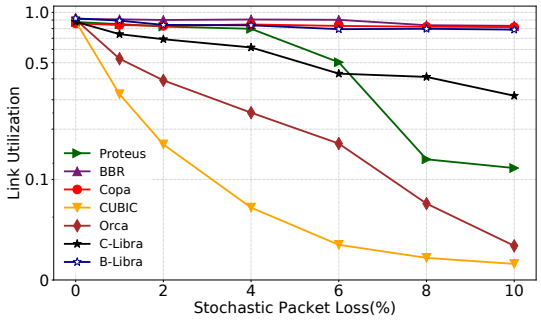


Figure 10: Impact of stochastic loss on performance.

5.2 Performance on Flexibility

We report Libra can provide a flexible trade-off between throughput and latency by changing the weights of variables. To validate it, we form five different utility functions: default, Th-1 (2× default α), Th-2 (3× default α), La-1 (2× default β), La-2 (3× default β), and use the same emulated networks as that in Fig. 7 to test their performance. As shown in Fig. 11(a) and Fig. 11(b), Libra can obtain the expected performance preferences and meet diverse application requirements. Specifically, Libra can surpass state-of-the-art CCAs in their strong points — for instance, Libra reaches a higher throughput than BBR when using throughput-oriented utility functions while achieving lower delay than Orca when changing to latency-aware utility functions. Then, we evaluate the aggressiveness, where Fig. 11(c) and Fig. 11(d) show that one Libra flow presents different degrees of aggressiveness when competing with one CUBIC flow. Libra obtain 48.4% $\sim$ 74.1% (using C-Libra) or 35.5% $\sim$ 49.6% (using B-Libra) bandwidth when we vary the weights on throughput and latency. Here 50% represents a fair position.

Remark 4: Our combined framework empowers flexibility. To give intuition, we snapshot Libra's trace in a control cycle where x_{cl} is around 45 Mbps and x_{rl} 50 Mbps. After applying them one by one in the evaluation stage, Libra collects the performance statistics ($45Mbps, 120ms, 0\%loss$) and ($50Mbps, 150ms, 5\%loss$), respectively. And the final rate decision depends on the weights of the utility function — x_{rl} will obtain a higher utility value with a throughput-oriented utility function, while x_{cl} will obtain a higher utility value with a delay-oriented one. In this way, Libra fine-grained tunes the performance preference according to application preferences, showing good flexibility.

5.3 Performance on Practicality

We report the performance of benchmark CCAs in terms of overhead, fairness, convergence properties, and the ability to provide safety assurance.

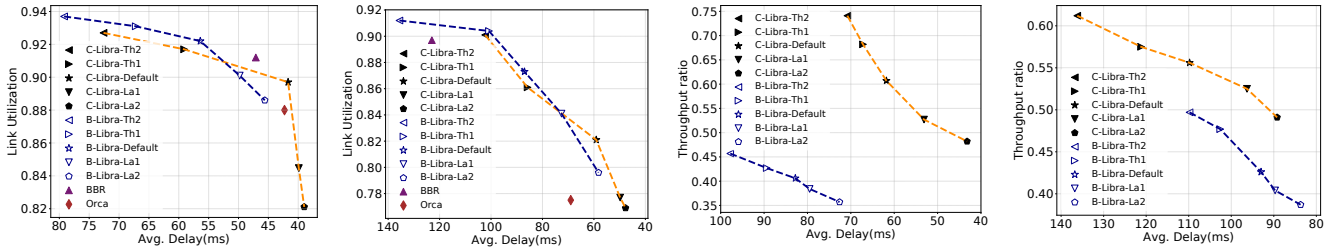
Overhead: Here we extend our prior experiments (Fig. 2(c)) by varying link capacity from 10 Mbps to 200 Mbps. We measure the average CPU utilization as the metric of overhead. For the CCAs that are implemented in the kernel, we report the CPU utilization of iperf [2]. Fig. 12 shows that Libra's overhead is consistently comparable with its underlying classic CCAs (CUBIC&BBR) while showing an average reduction of 47%, 54%, 59%, 79%, 84%, 92% over Orca, Clean-slate Libra, Modified RL, Indigo, Copa and Proteus, respectively.

Remark 5: The overhead of Libra, and of other CCAs that involve RL techniques, mainly comes from their DRL agents [3]. Compared with other CCAs using DRL agent (Orca, Clean-slate Libra, Modified RL, *etc.*), Libra has way lower overhead since its combined framework enables the costly DRL agent to work only in part of the control cycle (*i.e.*, exploration stage) to make candidate decisions.

Fairness and convergence: We evaluate the fairness and convergence property of Libra on a 48Mbps link with 100ms minimum RTT and 1BDP buffer. For inter-protocol fairness, we run two flows with the most popular CUBIC and the CCA under test, respectively. As for intra-protocol fairness, two senders using the same CCA are deployed to send traffic on the link. As shown in Fig. 13, both C-Libra and B-Libra have good inter-protocol fairness, achieving an over 98% Jain's fairness index, while pure learning-based CCAs such as Proteus, Aurora, and Modified RL perform poorly. Fig. 14 also indicates that Libra achieves improved intra-protocol fairness (around 99% Jain's fairness index) than pure learning-based CCAs.

For the convergence, we start three flows one by one (with an interval of 5 seconds) using the same CCA. Fig. 15 shows the dynamics of each flow's throughput over time, and the shadow area represents the link capacity. We quantify the experimental results shown in Fig. 15 to better argue our performance gains on convergence property. In Tab. 5, the convergence time is calculated as the time from the third flow's entry to the earliest time after which it maintains a stable sending rate (within ±25%) for 5 seconds. The stability is calculated as the standard deviation of throughput of the third flow after its convergence [12]. We also report the average throughput of the third flow after its convergence. We do not calculate these statistics for Modified RL since it cannot converge to a fair equilibrium in our experiments (Fig. 15). The results show that Libra can quickly converge to the equilibrium while showing good stability compared with the underlying classic CCA (*i.e.*, CUBIC and BBR). Other CCAs may present a long convergence time (*e.g.*, CUBIC and Proteus), an under-utilization equilibrium rate (*e.g.*, Indigo) or even an unfair share of the link capacity (*e.g.*, Mod. RL).

Remark 6: First, it is known that the goal of inter-protocol fairness (*i.e.*, when competing with CUBIC) is provably incompatible with the goals of achieving high efficiency and rapid ramp-up[7, 41]. Therefore, we do not expect Libra to outshine all the other CCAs on inter-protocol fairness, but to avoid starving CUBIC as other RL-based CCAs (*e.g.*, Aurora). Libra can achieve this goal perfectly. Besides, Libra's improvement over Modified RL in Fig. 13 highlights



(a) Performance of single flow on wired networks (b) Performance of single flow on cellular networks (c) Libra competes with CUBIC on wired networks (d) Libra competes with CUBIC on cellular networks

Figure 11: Libra's performance on flexibility.

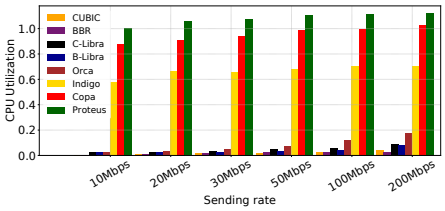


Figure 12: Overhead

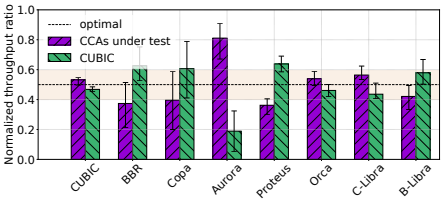


Figure 13: Inter-protocol fairness

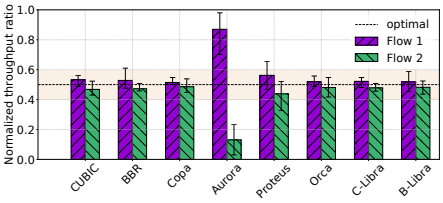


Figure 14: Intra-protocol fairness

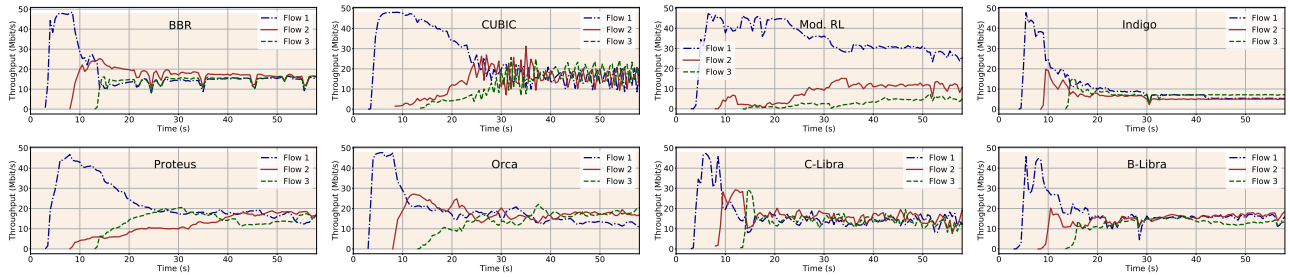


Figure 15: Convergence property of different CCAs

the advantage of combining classic CCAs. Second, Libra's intra-protocol fairness and convergence derive from both the utility function of Eq. 1 and the heavily coupled rate control algorithm (Alg. 1). Therefore, simply applying Eq. 1 (*e.g.*, Modified RL) to RL-based CCA, whose adjustments lack guarantees on convergence, cannot perform well on fairness and convergence (Fig. 14, Fig. 15). Besides, Libra's rapid convergence and reduced oscillation indicate another advantage — compared with online-learning approaches, Libra's combined framework and the wisdom of underlying CCAs help it to effectively identify the change of network conditions and quickly converge to the equilibrium point.

Safety assurance: To quantify the performance on providing safety assurance, we measure the average link utilization of Orca and Libra over 20 repeated experiments (*i.e.*, l_i where $i \in [0, 20]$), and then report the mean value, the range (*i.e.*, $l_{max} - l_{min}$) and the standard deviation of them in Tab. 6, where *#O*, *#C* and *#B* indicate Orca, C-Libra and B-Libra, respectively. We can observe that the link utilization of Libra only fluctuates in a small range (*i.e.*, 3.2%~11.7%), while that of Orca is highly variable (*i.e.*, 13.1%~28.8%). Besides, the lower standard deviation (*i.e.*, 0.17~0.52× than Orca) also shows that the performance of Libra is more reliable.

Table 5: Quantitative convergence property of different CCAs

	BBR	CUBIC	Mod. RL	Indigo
Conv. Time	6.2s	14.8s	-	5.4s
Thr. Deviation	1.81Mbps	5.97Mbps	-	1.09Mbps
Avg. Throughput	16.0Mbps	15.9Mbps	-	8.2Mbps
	Proteus	**Orca**	**C-Libra**	**B-Libra**
Conv. Time	17.2s	7.8s	3.6s	4.1s
Thr. Deviation	2.51Mbps	3.29Mbps	2.17Mbps	1.97Mbps
Avg. Throughput	16.1Mbps	16.0Mbps	15.8Mbps	16.0Mbps

Remark 7: The property of safety assurance focuses on the performance fluctuation over repeated experiments. Learning-based CCAs always show a highly variable performance over multiple runs due to the stochasticity in the learning procedure [17]. Libra can address this issue perfectly — by carefully evaluating the decisions from underlying CCAs, it avoids the unexpected decisions

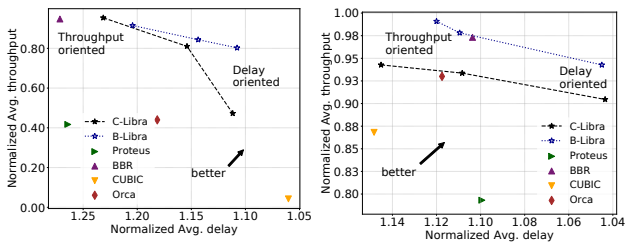
Table 6: Statistics of the link utilization of 20 trials in different networks

	Wired#1 (24Mbps)	Wired#2 (48Mbps)	LTE#1 (Stationary)	LTE#2 (Moving)
Mean#O	0.891	0.908	0.813	0.876
Mean#C	0.886	0.912	0.897	0.900
Mean#B	0.903	0.917	0.871	0.883
Range#O	0.135	0.249	0.288	0.131
Range#C	0.052	0.080	0.117	0.096
Range#B	0.032	0.044	0.092	0.079
Std dev.#O	0.043	0.098	0.103	0.050
Std dev.#C	0.016	0.021	0.028	0.026
Std dev.#B	0.011	0.017	0.021	0.019

from the DRL agent while leveraging the wisdom of classic CCAs to achieve safety assurance.

Remark 8: We find that the properties of Libra's stochastic loss resilience and inter-protocol fairness are probably changed with different underlying classic CCAs. However, the provable convergence, predictable behaviors and negligible overhead are always achieved by Libra due to the general advantages of the classic CCAs.

5.4 Performance on Live Internet



(a) Inter-continental scenario (b) Intra-continental scenario

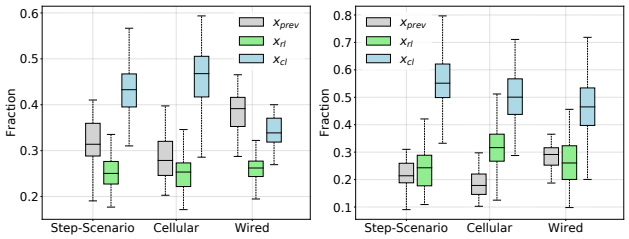
Figure 16: Normalized performance on live Internet

To investigate Libra's performance in live Internet, we conduct experiments on Amazon EC2 platforms. We separately measure the performance of each CCA in intra- and inter-continental scenarios, which can cover a wide range of network conditions. We average the normalized performance of each CCA and the results are shown in Fig. 16. First, compared with the results reported in Fig. 7, we can see that Orca and CUBIC significantly drop throughput in our inter-continental scenarios due to the complexity of these scenarios, *e.g.*, higher stochastic loss rate, different queue management schemes and traffic shaping schemes unknown to the end-points [2]. C-Libra shows a flexible performance preference on throughput and delay with different utility functions — it can obtain a 6% higher throughput (with throughput-oriented version) or 14.4% lower delay (with delay-oriented version) than BBR. B-Libra performs a similar throughput but 5.6%~14.8% lower delay than BBR. In the case of the intra-continental scenarios, Libra also shows a high performance on throughput and delay, which highlights the good adaptability of Libra over different network conditions.

5.5 Deep Dive to the Improvements

The fraction of applied times for each candidate rate: To give a deep dive into the benefits of Libra's combined framework, we separately test C-Libra and B-Libra in several scenarios for 50 times and plot the fraction of applied times among x_{prev} (the previous decision), x_{rl} (learning-based CCA's sending rate) and x_{cl} (classic CCA's sending rate) in Fig. 17. A higher ratio here indicates that Libra is more likely to take advantage of this kind of rate decisions.

As expected, every kind of rate decisions plays an important role in adjusting the sending rate. On average, when using CUBIC, the applied decisions coming from x_{prev}, x_{rl} and x_{cl} account for 32%, 26% and 42% of the total number of control cycles, respectively. The fractions are 23%, 27% and 50% when using BBR. We find that Libra is more helpful to CUBIC in wired networks (the control cycles applying x_{cl} is significantly less than that in other scenarios) since it avoids repeated cycles of filling the buffer and the rate reduction. When using BBR, Libra tends to benefit it in cellular networks by accelerating the bandwidth probing procedure using the advice from learning-based CCAs.



(a) C-Libra (b) B-Libra

Figure 17: Fraction of applied times for x_{pre}, x_{rl} and x_{cl}

Remark 9: From Fig. 17, we can observe that no existing CCA can perform the best all the time. This reveals the source of Libra's performance gains to some extent. That is, Libra can outperform each of the underlying CCAs by periodically selecting the better action from them. The improvement becomes significant especially under some complex scenarios that a single underlying CCA cannot perform well.

Compare Libra with ideal combination versions: To verify the effectiveness of Libra's combination mechanism, we conduct experiments to compare Libra with ideal combination versions — C-Ideal and B-Ideal. To obtain C-Ideal, we run CUBIC and Clean-Slate Libra individually under the same emulated networks, and then for each time step, the behaviors from the two with higher utility values generate it. Hence, C-Ideal is essentially an offline combined version of CUBIC and Clean-slate Libra, which takes the one with higher utility value over the time horizon as its own part. In the same way, we can generate B-Ideal. Note that as an offline combination, C-Ideal and B-ideal do not involve the interaction between the classic CCA and the learning-based CCA. To clarify the source of performance gains, we scale the utility to a normalized range ([0,1]) and present the results in Fig. 18. We can observe that both C-Libra and B-Libra obtain high utilities most of the time, approaching or even surpassing their ideal combined versions (*e.g.*, 28~30s and 42~50s for C-Libra, 15~23s for B-Libra).

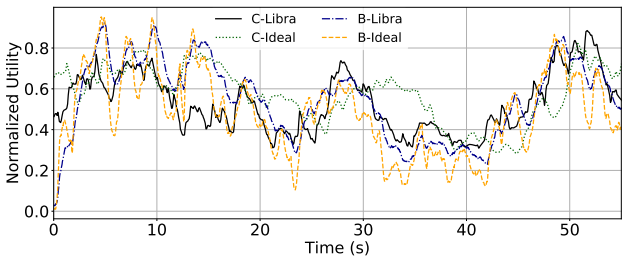


Figure 18: The utility comparisons in a cellular network.

Remark 10: The experiment results demonstrate that Libra's combined framework wouldn't degrade the performance advantages of each underlying CCA, while it can even gain better performance. We argue that the improvement comes from the complementary advantages and interaction between two underlying CCAs. That is, compared with the ideal combination versions, Libra enables two underlying CCAs to interact with each other periodically — one CCA may reset the other CCA's sending rate via Libra's evaluation stage (*e.g.*, the sending rate with higher utility value is preferred). Though the performance of Libra is good in general, we should also pay attention to the cases that Libra sometimes earns lower utility values than that of the ideal version (12∼18s and 31∼37s for C-Libra, 5∼7s for B-Libra). We leave further optimizations on Libra's framework to future work.

6 RELATED WORK

Classic CCAs: Classic CCAs usually take the loss or RTT variations as indicators of congestion and react to them with pre-defined policies [5, 6, 8, 16, 36]. However, loss-based CCAs suffer the notorious bufferbloat problem [3, 39] and wrong reduction under stochastic losses [8]. For delay-based CCAs, although stabilization and lower delay are offered, a series of issues such as severe unfairness may happen when competing with loss-based CCAs. Besides, classic CCAs always consider unique features and characteristics for target networks during the design [8], and thus cannot excel in diverse network conditions [9].

Learning-based CCAs: Recently, a great deal of effort has been made to abandon the hardwired mapping between certain events and certain actions, and allow machines to automatically generate the best policy by themselves. Some designs, such as Remy [33], Aurora [20], Eagle [13], and Indigo [38] train a control model offline with parameterized emulations. Others [12, 25] take online-learning approaches by using gradient ascent algorithms to optimize the utility function. However, all these CCAs confront practical problems in terms of overhead, fairness, or convergence [2]. Besides, Rotman et al. [28] raise safety concerns about pure learning-based approaches, while their solution of running multiple RL agents to infer the unseen scenario in real time inevitably leads to high overhead.

Combined CCAs: Combining the wisdom of different CCAs to achieve performance improvements is attractive. Some works adopt a combination of different congestion signals (*i.e.*, ECN and end-to-end delay in GEMINI [42]) or different policies to generate cwnd (*e.g.*, CTCP [32]). Recently, Orca [2] combines RL-based CCA with CUBIC by periodically using the decisions from the DRL agent to adjust the rate of CUBIC. Rein [9] switches different CCAs midway according to the run-time environments. Our work is partly inspired by the above work but more general and practical in both the designs and the goals.

7 DISCUSSION

How to choose Libra's parameters? Based on the underlying classic CCA, the guidance provided by previous works and the observations from our experimental results, we accordingly set the default parameters for Libra. First, the duration of the exploration stage and the threshold (th) in Libra are tuned based on the underlying classic CCAs (Sec. 4.3). Second, based on the prior choices (0.5 ∼ 3 RTTs) [2, 13, 23, 25], we experimentally evaluate the impact of EI's duration in dynamic networks and finally set it to 0.5 estimated RTT. The key observation is that evaluating an improper decision for a long time harms Libra's link utilization (Fig. 19). The duration of the exploitation stage is set to accommodate the ACKs of the packets that are sent in the evaluation stage while balancing the desires of fully exploiting the good decisions against the goal of achieving high responsiveness in highly variable networks. Note that the appropriate parameter settings for CUBIC and BBR can be extended to a wide range of classic CCAs (*e.g.*, Westwood, Illinois). Due to the detailed guidelines and low parameter sensitivity (Appendix B), we believe that it is also practical to choose parameters when combining other underlying classic CCAs.

What if we apply Libra to other networks? Currently, Libra is well tested over wired and cellular networks. Due to the good adaptability (Sec. 5.1), Libra should handle many key characteristics in other networks including long RTT and high stochastic loss rate in satellite networks [23], the abrupt fluctuation on available link capacity in 5G scenarios [35]. Besides, Libra can replace its classic counterparts with classic CCAs that are designed for specific networks [4, 22, 35] to leverage new properties (*e.g.*, ECN marking, hardware timestamp, 5G's network slicing) and address more challenges (*e.g.*, incast and extremely low RTT in datacenters). We leave them for future work.

8 CONCLUSION

In this paper, we present Libra, a combined congestion control framework to complement the advantages of both classic CCAs and learning-based CCAs. Libra can adapt to diverse network conditions and adjust performance preferences according to application demands. Extensive experiments on live internet and Pantheon clarify the source of performance gains of Libra's combination mechanism and demonstrate that Libra can achieve our design goals of adaptability, flexibility and practicality.

ACKNOWLEDGMENTS

We would like to thank our shepherd and anonymous reviewers for their valuable comments. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province (2020B0101390001), China NSF grants (62172206, 61972254, 61802172), China NSF of Jiangsu Province (BK20201248) and Open Fund of PDL (WDZC20205500109).

REFERENCES

[1] Soheil Abbasloo, Yang Xu, and H. Jonathan Chao. 2019. C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements. *IEEE J. Sel. Areas Commun.* 37, 4 (2019), 918–932. https://doi.org/10.1109/JSAC.2019.2898758

[2] 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 ACM SIGCOMM*, Henning Schulzrinne and Vishal Misra (Eds.). ACM, 632–647. https://doi.org/10.1145/3387514.3405892

[3] 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 J. Sel. Areas Commun.* 39, 1 (2021), 265–279. https://doi.org/10.1109/JSAC.2020.3036958

[4] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of ACM SIGCOMM*, Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker (Eds.). ACM, 63–74. https://doi.org/10.1145/1851182.1851192

[5] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In *Proceedings of USENIX NSDI*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 329–342. https://www.usenix.org/conference/nsdi18/presentation/arun

[6] Lawrence S. Brakmo and Larry L. Peterson. 1995. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE J. Sel. Areas Commun.* 13, 8 (1995), 1465–1480. https://doi.org/10.1109/49.464716

[7] Lloyd Brown, Ganesh Ananthanarayanan, Ethan Katz-Bassett, Arvind Krishnamurthy, Sylvia Ratnasamy, Michael Schapira, and Scott Shenker. 2020. On the Future of Congestion Control for the Public Internet. In *Proceedings of ACM HotNets*, Ben Zhao, Heather Zheng, Harsha V. Madhyastha, and Venkat N. Padmanabhan (Eds.). ACM, 30–37. https://doi.org/10.1145/3422604.3425939

[8] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66. https://doi.org/10.1145/3009824

[9] Kefan Chen, Danfeng Shan, Xiaohui Luo, Tong Zhang, Yajun Yang, and Fengyuan Ren. 2020. One Rein to Rule Them All: A Framework for Datacenter-to-User Congestion Control. In *Proceedings of ACM APNet*. ACM, 44–51. https://doi.org/10.1145/3411029.3411036

[10] A. V. Chernov. 2019. On Some Approaches to Find Nash Equilibrium in Concave Games. *Autom. Remote. Control.* 80, 5 (2019), 964–988. https://doi.org/10.1134/S0005117919050138

[11] Mo Dong, Qingxi Li, Doron Zarchy, Philip Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proceedings of USENIX NSDI*. USENIX Association, 395–408. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong

[12] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *Proceedings of USENIX NSDI*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 343–356. https://www.usenix.org/conference/nsdi18/presentation/dong

[13] Salma Emara, Baochun Li, and Yanjiao Chen. 2020. Eagle: Refining Congestion Control by Learning from the Experts. In *Proceedings of IEEE INFOCOM*. IEEE, 676–685. https://doi.org/10.1109/INFOCOM41043.2020.9155250

[14] Eyal Even-Dar, Yishay Mansour, and Uri Nadav. 2009. On the convergence of regret minimization dynamics in concave games. In *Proceedings of ACM STOC*, Michael Mitzenmacher (Ed.). ACM, 523–532. https://doi.org/10.1145/1536414.1536486

[15] Sally Floyd and Thomas R. Henderson. 1999. The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC* 2582 (1999), 1–12.

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Oper. Syst. Rev.* 42, 5 (2008), 64–74. https://doi.org/10.1145/1400097.1400105

[17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep Reinforcement Learning That Matters. In *Proceedings of AAAI*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 3207–3214. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16669

[18] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines. (2018).

[19] Yannis E. Ioannidis and Eugene Wong. 1987. Query Optimization by Simulated Annealing. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data*, Umeshwar Dayal and Irving L. Traiger (Eds.). ACM Press, 9–22. https://doi.org/10.1145/38713.38722

[20] Nathan Jay, Noga H. Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proceedings of PMLR ICML*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 3050–3059. http://proceedings.mlr.press/v97/jay19a.html

[21] Yiming Kong, Hui Zang, and Xiaoli Ma. 2018. Improving TCP Congestion Control with Machine Intelligence. In *Proceedings of the 2018 Workshop on Network Meets AI & ML, NetAI@SIGCOMM*. ACM, 60–66. https://doi.org/10.1145/3229543.3229550

[22] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of ACM SIGCOMM*, Henning Schulzrinne and Vishal Misra (Eds.). ACM, 514–528. https://doi.org/10.1145/3387514.3406591

[23] Xu Li, Feilong Tang, Jiacheng Liu, Laurence T. Yang, Luoyi Fu, and Long Chen. 2021. AUTO: Adaptive Congestion Control Based on Multi-Objective Reinforcement Learning for the Satellite-Ground Integrated Network. In *Proceedings of USENIX ATC*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 611–624. https://www.usenix.org/conference/atc21/presentation/li-xu

[24] Yuxi Li. 2017. Deep Reinforcement Learning: An Overview. *CoRR* abs/1701.07274 (2017). arXiv:1701.07274 http://arxiv.org/abs/1701.07274

[25] Tong Meng, Neta Rozen Schiff, Philip Brighten Godfrey, and Michael Schapira. 2020. PCC Proteus: Scavenger Transport And Beyond. In *Proceedings of ACM SIGCOMM*, Henning Schulzrinne and Vishal Misra (Eds.). ACM, 615–631. https://doi.org/10.1145/3387514.3405891

[26] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of USENIX ATC*, Shan Lu and Erik Riedel (Eds.). USENIX Association, 417–429. https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[27] Kathleen M. Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (2012), 42–50. https://doi.org/10.1145/2209249.2209264

[28] Noga H. Rotman, Michael Schapira, and Aviv Tamar. 2020. Online Safety Assurance for Learning-Augmented Systems. In *Proceedings of ACM HotNets*, Ben Zhao, Heather Zheng, Harsha V. Madhyastha, and Venkat N. Padmanabhan (Eds.). ACM, 88–95. https://doi.org/10.1145/3422604.3425940

[29] Alessio Sacco, Matteo Flocco, Flavio Esposito, and Guido Marchetto. 2021. Owl: Congestion Control with Partially Invisible Networks via Reinforcement Learning. In *Proceedings of IEEE INFOCOM*. IEEE, 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488851

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[31] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. In *Proceedings of ACM SIGCOMM*, Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy (Eds.). ACM, 479–490. https://doi.org/10.1145/2619239.2626324

[32] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. 2006. A Compound TCP Approach for High-Speed and Long Distance Networks. In *Proceedings of IEEE INFOCOM*. IEEE. https://doi.org/10.1109/INFOCOM.2006.188

[33] Keith Winstein and Hari Balakrishnan. 2013. TCP ex machina: computer-generated congestion control. In *Proceedings of ACM SIGCOMM*, Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan (Eds.). ACM, 123–134. https://doi.org/10.1145/2486001.2486020

[34] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of USENIX NSDI*, Nick Feamster and Jeffrey C. Mogul (Eds.). USENIX Association, 459–471. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/winstein

[35] Yaxiong Xie, Fan Yi, and Kyle Jamieson. 2020. Pbe-CC: Congestion Control via Endpoint-Centric, Physical-Layer Bandwidth Measurements. In *Proceedings of ACM SIGCOMM*, Henning Schulzrinne and Vishal Misra (Eds.). ACM, 451–464. https://doi.org/10.1145/3387514.3405880

[36] Lisong Xu, Khaled Harfoush, and Injong Rhee. 2004. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *Proceedings of IEEE INFOCOM*. IEEE, 2514–2524. https://doi.org/10.1109/INFCOM.2004.1354672

[37] Zhiyuan Xu, Jian Tang, Chengxiang Yin, Yanzhi Wang, and Guoliang Xue. 2019. Experience-Driven Congestion Control: When Multi-Path TCP Meets Deep Reinforcement Learning. *IEEE J. Sel. Areas Commun.* 37, 6 (2019), 1325–1336. https://doi.org/10.1109/JSAC.2019.2904358

[38] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Alexander Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *Proceedings of USENIX ATC*, Haryadi S. Gunawi and Benjamin Reed (Eds.). USENIX Association, 731–743. https://www.usenix.org/conference/atc18/presentation/yan-francis

[39] Jiancheng Ye, Ka-Cheong Leung, Victor O. K. Li, and Steven H. Low. 2018. Combating Bufferbloat in Multi-Bottleneck Networks: Equilibrium, Stability, and Algorithms. In *Proceedings of IEEE INFOCOM*. IEEE, 648–656. https://doi.org/10.1109/INFOCOM.2018.8486251

[40] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular

Networks. In *Proceedings of ACM SIGCOMM*, Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye (Eds.). ACM, 509–522. https://doi.org/10.1145/2785956.2787498

[41] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. 2017. An Axiomatic Approach to Congestion Control. In *Proceedings of ACM HotNets*, Sujata Banerjee, Brad Karp, and Michael Walfish (Eds.). ACM, 115–121. https://doi.org/10.1145/3152434.3152445

[42] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. 2019. Congestion Control for Cross-Datacenter Networks. In *Proceedings of IEEE ICNP*. IEEE, 1–12. https://doi.org/10.1109/ICNP.2019.8888042

A PROOF OF THE CONVERGENCE AND FAIRNESS

Our proof on convergence and fairness mainly consists of the following two parts. The first part proves the existence and uniqueness of the fair equilibrium based on the choice of the utility function (Eq. 1). Then we prove that Libra's rate control algorithm (Alg. 1) enables Libra to converge to this equilibrium.

A.1 Notations

Consider n Libra senders compete on a single bottleneck with capacity C, we assume x_i is the sending rate of i_{th} sender and $S = \sum_{i=0}^{n} x_i$ is the total sending rate of all senders. Under the droptail queue, we can derive that (1) if $S \geq C$, the loss rate $L = \left(1 - \frac{C}{S}\right)$; (2) the RTT gradient $\frac{d(RTT_i)}{dt} = \frac{S-C}{C}$. Accordingly, the utility function (Eq. 1) can be described as:

$$u(x_i) = \alpha \cdot x_i^t - \beta \cdot x_i \cdot \max\left\{0, \frac{S-C}{C}\right\} - \gamma \cdot x_i \cdot \left(1 - \frac{C}{S}\right)$$

A.2 Existence and uniqueness of equilibrium

Based on the notations above, we formulate this interactive model as a non-cooperative game G, in which the Libra senders represent the players, and the utility function specifies the payoff of strategies. In general, we have

Lemma A.1. *[25] There is no equilibrium when $S < C$, which means that for any equilibrium the total sending rate S should be greater than or equal to the bottleneck capacity C.*

Specifically, for the utility function we used, we obtain

Lemma A.2. *Given the utility function as Eq. 1, there exists a unique Nash equilibrium for the non-cooperative game G when $S \geq C$.*

Proof. We prove the existence and uniqueness of Nash equilibrium by showing that G ($S \geq C$) is strictly socially concave [14]. That is, the game G should meet the following properties as follows: (1) each sender i's utility function is strictly concave with respect to its rate x_i; (2) each sender i's utility function is convex with respect to the other's sending rate $x_{-i} = \sum_{j \neq i} x_j$; (3) the total utility value $U = \sum_i u(x_i)$ is concave. First, we calculate the second derivative of $u(x_i)$ as

$$\frac{\partial^2 u(x_i)}{\partial (x_i)^2} = \alpha t(t-1) x_i^{t-2}$$

If $0 < t < 1$ and $\alpha > 0$, then the second derivative of $u(x_i)$ is always negative, and thus $u(x_i)$ is concave with respect to x_i. Second, since $u(x_i) = \alpha \cdot x_i^t - \beta \cdot x_i \frac{x_i + x_{-i} - C}{C} - \gamma \left(1 - \frac{C}{x_i + x_{-i}}\right)$ and its second derivative

$$\frac{\partial^2 u(x_i)}{\partial (x_{-i})^2} = c x_i \frac{C}{S^3}$$

is positive. Hence we can conclude that each sender i's utility function is convex with respect to the other's sending rate x_{-i}. Finally, the sum of the utility values is

$$U = \sum_i u(x_i) = \alpha \sum_i x_i^t - \beta S \frac{S-C}{C} - \gamma S \left(1 - \frac{C}{S}\right).$$

And its second derivative can be calculated by

$$\frac{\partial^2 U(S)}{\partial S^2} = \frac{\partial^2}{\partial S^2}\left(\alpha \sum_i x_i^t - \beta S \frac{S-C}{C} - \gamma S \left(1 - \frac{C}{S}\right)\right)$$
$$= \alpha \frac{\partial^2 \sum_i x_i^t}{\partial S^2} - \frac{2\beta}{C}.$$

Since $0 < t < 1$, $\alpha > 0$, and x_i^t is concave, $\alpha \sum_i x_i^t$ is concave as well. Subsequently, we can obtain that $\frac{\partial^2 \sum_i x_i^t}{\partial S^2} < 0$. For the second term, remember $\beta > 0$ and we have $-\frac{2\beta}{C} < 0$. Therefore, $\frac{\partial^2 U(S)}{\partial S^2} < 0$ and the total utility value $U(S)$ is concave. □

Lemma A.3. *Given the utility function as Eq. 1, the unique equilibrium of G is fair, where $\overline{x_1} = \overline{x_2} = \cdots = \overline{x_n}$ and $\sum_{i=1}^{n} x_i \geq C$.*

Proof. First, from Lemma A.1 and Lemma A.2, we can derive that the game G has a unique Nash equilibrium. Next, suppose that a sending rate x_i is different from another rate x_j in the equilibrium, then we can obtain a different equilibrium by simply exchanging these two rates, which contradict the Lemma A.2. Therefore, the unique equilibrium of the game G is fair. Last, suppose that $\sum_{i=1}^{n} \overline{x} < C$. In such a case, each Libra sender can increase its sending rate x_i without incurring the packet loss, which leads to a higher utility value. This contradicts the definition of the Nash equilibrium and concludes our proof.

□

A.3 Ability to converge to the fair equilibrium.

The utility function ensures the existence and uniqueness of the equilibrium. Now we begin to analyze how Libra can reach this equilibrium. Without loss of generality, we consider that two Libra senders (say i and j) compete on a single bottleneck, where the equilibrium rate is $\overline{x}$ and the underlying classic CCA is CUBIC. Note that any classic CCA with provable convergence can also make our Libra reach the equilibrium point.

Lemma A.4. *The rate control algorithm (Alg. 1) enables Libra to converge to the fair equilibrium.*

Proof. We prove Lemma A.4 by showing that in all conditions Libra can adjust its sending rate towards the equilibrium rate.
(i): $S < C$. We first prove that Libra senders will continue to increase their rates x_i and x_j in the under-utilization case (*i.e.*, $x_i < x_j < \overline{x}$ or $x_i < \overline{x} < x_j$). This is because, before fully utilizing the link capacity, the underlying classic CCA (*i.e.*, CUBIC) will increase its rate x_{cl}. In such cases, a higher rate between x_{cl} and x_{rl} is preferred due to the higher utility value obtained. This ensures a rapid increase of Libra's base sending rate in the presence of available capacity.
(ii): $S > C$. Under this condition, we claim that Libra can quickly decrease its sending rate and move to the equilibrium point. First, the decisions from classic CCAs always drive Libra towards a lower sending rate (*i.e.*, $x_i \cdot \eta$ where $\eta < 1$) after detecting congestion. At

the same time, RL-based CCA observes a similar state vector and outputs the same decision with its MIMD action space (*i.e.*, $x_i^t \cdot \theta$ or $x_j^t \cdot \theta$). We claim that the difference in the next control cycle $t+1$ between the base sending rate x_i^{t+1} and x_j^{t+1} is smaller than or equal to the current difference $|x_i^t - x_j^t|$. Here we consider three possible adjustments of Libra: (1) Libra chooses decisions of the classic CCA, we can derive $x_i^{t+1} - x_j^{t+1} = x_i^t \cdot (1-\eta) - x_j^t \cdot (1-\eta) = (1-\eta) \cdot (x_i^t - x_j^t)$ and $(1-\eta)(x_i^t - x_j^t) < x_i^t - x_j^t$, where $0 < \eta < 1$; (2) Libra chooses decisions of the RL-based CCA, we can derive $x_i^{t+1} - x_j^{t+1} = x_i^t \cdot \theta - x_j^t \cdot \theta = \theta \cdot (x_i^t - x_j^t)$, and also $\theta \cdot (x_i^t - x_j^t) < (x_i^t - x_j^t)$, where $0 < \theta < 1$. When $\theta > 1$, the decisions of the RL-based CCA will gain a lower utility value and thus cannot be selected. (3) remain the same sending rate as before, and then the difference remains unchanged as well. The decision that obtains a higher utility value is consistent among all senders. As a result, it is impossible to select one decision from the RL-based CCA for the sender i and the other decision from the classic CCA for the sender j.

(iii): $S = C$. Suppose that $x_i \neq x_j$. Under this condition, each Libra sender may obtain a higher utility value and turn to case (ii) by increasing its sending rate. Otherwise, the current sending rates of x_i and x_j form an equilibrium, which is a contradiction to Lemma A.3.

□

B PARAMETER SENSIBILITY

We change the duration of each stage and the threshold th_1 under the same emulated networks as that in Fig. 7. We report the performance in Fig. 19 and Tab. 7. The experimental results well match our analyses that: (1) longer exploration and exploitation stages result in significant degradation of link utilization (4.4% on average) in highly varying cellular links. (2) evaluating an improper candidate rate for a long time (*i.e.*, tune EI from 0.5 to 1 RTT) harms the link utilization. (3) Libra shows low sensitivity to different parameters and our default settings are appropriate.

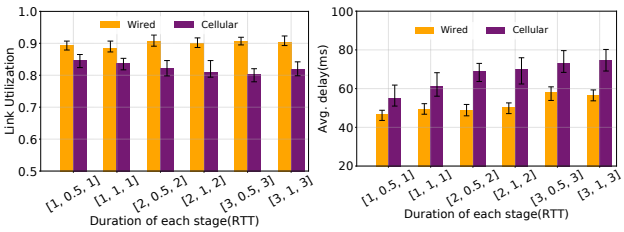


Figure 19: Avg. Performance of C-Libra under different duration of stages

When enlarging the duration, the performance degradation in a highly varying scenario (*e.g.*, cellular networks) is attributed to its slower reaction to network dynamics. But for the scenario with more stable network characteristics (*e.g.*, our emulated wired scenarios), a longer duration is more appropriate, since it can well exploit the good decisions with less unnecessary attempts to explore.

Table 7: Avg. Performance of C-Libra under different switching thresholds

Configuration	Link utilization	Avg. delay(ms)
Wired-0.1×	87.7%	43.2
Wired-0.2×	88.6%	43.2
Wired-0.3×	88.9%	43.5
Wired-0.4×	89.0%	44.4
Cellular-0.1×	81.7%	67.0
Cellular-0.2×	82.7%	61.8
Cellular-0.3×	83.6%	57.0
Cellular-0.4×	83.1%	56.3