Bridging the Gap between QoE and QoS in Congestion Control: A Large-scale Mobile Web Service Perspective

Jia Zhang, *Tsinghua University, Zhongguancun Laboratory, Beijing National Research Center for Information Science and Technology;* Yixuan Zhang, *Tsinghua University, Beijing National Research Center for Information Science and Technology;* Enhuan Dong, *Tsinghua University, Quan Cheng Laboratory, Beijing National Research Center for Information Science and Technology;* Yan Zhang, Shaorui Ren, and Zili Meng, *Tsinghua University, Beijing National Research Center for Information Science and Technology;* Mingwei Xu, *Tsinghua University, Quan Cheng Laboratory, Beijing National Research Center for Information Science and Technology;* Xiaotian Li, Zongzhi Hou, and Zhicheng Yang, *Meituan Inc.;* Xiaoming Fu, *University of Goettingen*

https://www.usenix.org/conference/atc23/presentation/zhang-jia

This paper is included in the Proceedings of the 2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

Open access to the Proceedings of the 2023 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Bridging the Gap between QoE and QoS in Congestion Control: A Large-scale Mobile Web Service Perspective

Jia Zhang[1,2,4], *Yixuan Zhang*[1,4], *Enhuan Dong*[1,3,4], *Yan Zhang*[1,4], *Shaorui Ren*[1,4], *Zili Meng*[1,4], *Mingwei Xu*[1,3,4], *Xiaotian Li*[5], *Zongzhi Hou*[5], *Zhicheng Yang*[5], *Xiaoming Fu*[6]
[1]*Tsinghua University,* [2]*Zhongguancun Laboratory,* [3]*Quan Cheng Laboratory,*
[4]*Beijing National Research Center for Information Science and Technology,*
[5]*Meituan Inc.,* [6]*University of Goettingen*

Abstract

To improve the user experience of mobile web services, various congestion control algorithms (CCAs) have been proposed, yet the performance of the application is still unsatisfactory. We argue that the suboptimal performance comes from the gap between what the application needs (i.e., Quality of Experience (QoE)) and what the current CCA is optimizing (i.e., Quality of Service (QoS)). However, optimizing QoE for CCAs is extremely challenging due to the convoluted relationship and mismatched timescale between QoE and QoS. To bridge the gap between QoE and QoS for CCAs, we propose Floo, a new QoE-oriented congestion control selection mechanism, as a shim layer between CCAs and applications to address the challenges above. Floo targets request completion time as QoE, and conveys the optimization goal of QoE to CCAs by always selecting the most appropriate CCA in the runtime. Floo further adopts reinforcement learning to capture the complexity in CCA selection and supports smooth CCA switching during transmission. We implement Floo in a popular mobile web service application online. Through extensive experiments in production environments and on various locally emulated network conditions, we demonstrate that Floo improves QoE by about 14.3% to 52.7%.

1 Introduction

Last decade has witnessed a dramatic increase in the use of mobile web services. The latest statistics demonstrate that more than 60% of global Internet users access web services with mobile devices [4]. Mobile web services are built on the transport layer, which typically employs a congestion control algorithm (CCA) that determines the data sending behavior and significantly affects the user experience. However, unsatisfactory performance of mobile web services has still been reported [40, 43, 49]. A recent measurement in 2020 shows that some web service users are still suffering from multi seconds request completion time [54], which also corroborates our observation in our web service in production (§5.3).

Our observation is that the key issue of the unsatisfactory performance in mobile web services is the mismatch between *what CCAs are optimizing* and *what the applications need*. Normally, CCAs optimize the Quality of Service (QoS) metrics describing the transport layer protocol delivery capabilities, i.e., delay, throughput, loss rate [22, 31], or a combination of these metrics [7, 19], in different network conditions. However, the applications do not really need an optimized QoS. Instead, they need a high quality of experience (QoE) for users. Specifically, for mobile web services, they have no idea what is *throughput* but only care about *request completion time (RCT)*[1]. The mismatch between QoE and QoS for current CCAs leads to their suboptimal performance.

However, optimizing QoE for CCAs is extremely challenging due to the following reasons. First, in terms of causality, the relationship between QoE and QoS is convoluted. For example, for small requests in web services, the increase of RTT will result in the degradation of the RCT. However, for large bulk requests, the effect of the increase of RTT on RCT is negligible. Therefore, without clearly understanding the relationship between QoE and QoS, directly optimizing the QoS may not be able to improve the QoE for the application. Second, in terms of timeliness, network conditions in QoS such as RTT and throughput are usually measured on a fine timescale (e.g., per packet). With this, CCAs can also make decisions on a short timescale. However, QoEs are usually perceived in a much longer timescale. RCTs in web services can only be calculated after the request completes, which usually takes seconds, during which, the CCA has already made numerous decisions. Thus, it is challenging for CCA to know which decisions are right and further correct its decisions (§2.1).

Our insight in this paper is not to directly optimize the CCA

[0]Jia and Yan are with Department of Computer Science and Technology, Tsinghua University. Yixuan, Enhuan and Zili are with Institute for Network Sciences and Cyberspace, Tsinghua University. Shaorui is with Department of Electronic Engineering, Tsinghua University. Mingwei is with Department of Computer Science and Technology and Institute for Network Sciences and Cyberspace, Tsinghua University. Xiaoming is with Institute of Computer Science, University of Goettingen.

[1]There are various QoE metrics of mobile web service, such as page load time. In this paper, we focus on the request completion time, i.e., the time interval between the request sent and the response fully received.

itself, but to introduce a shim layer between the application layer and transport layer to *select* the appropriate CCA for a better QoE. After decades of evolution of CCA, although there may not exist one CCA to fit all scenarios, we believe that there should always be at least one CCA that behaves well in a certain scenario. Selecting the CCA addresses the challenges above in two ways: First, instead of blindly optimizing those low-level instructions for CCAs with QoS, we could select the appropriate CCA based on the QoE. Second, we can perform the CCA selection at the same or longer timescale to fully and accurately utilize the information from QoE. Therefore, if we could always select and switch to the best CCA in the runtime, we will have the QoE directly optimized (§2.2).

However, it is challenging to propose a CCA selection mechanism for large-scale mobile web service due to the following reasons (§2.3).

- **Generating an optimal CCA selection policy is challenging.** The CCA selection policy, or in other words, mapping from the observed network conditions and application QoE metrics to the appropriate CCAs, is complicated. (1) The mobile network conditions can fluctuate, and are not easy to capture from the metrics observed on endpoints. (2) It is very difficult to model and characterize the CCAs, especially the recent proposed complicated CCAs [9, 19].
- **Switching between CCAs is nontrivial.** Different CCAs maintain different states. For example, BBR maintains the maximum delivery rate in the last 8 RTTs, but Cubic does not. If we need to switch between CCAs in the runtime, we should handle the states for a seamless switch carefully. Otherwise, if each CCA starts with the slow start, the QoE might be severely impaired during each switching.

To address the above challenges and provide better performance to the real applications, we propose Floo, a QoE-oriented mechanism for congestion control selection in large-scale mobile web services. Our key ideas are (1) to design *a QoE-oriented CCA selection mechanism*, and (2) to support *seamless CCA switching* during transmission. To turn our ideas into reality, we design several building blocks in Floo. First, we propose a reinforcement learning (RL)-based framework (to understand CCAs) that uses QoE as the selection criterion, and carefully selects both transport layer and application layer metrics (against network dynamic) to be jointly used in CCA selection (§3.2 and §3.3). Second, we devise a CCA switching mechanism to ensure the smoothness of switching by migrating the CCA phases and variables. The switching mechanism can be applied to traditional non-learning CCAs, and it is implemented with multiple classical CCAs in this paper (§3.4). Briefly speaking, Floo selects the optimal CCA for each connection according to QoE, and switches to a new, better CCA when the network condition changes (§3.1).

We implement Floo atop QUIC in the production environment of one Meituan's popular mobile web service application, Dianping, with O(10M) daily active users (§4.1). To make Floo work for real application scenarios, we collect real-world application traces for 14 days, including 35 million request logs. The traces are employed for analysis and training to reduce the gap between emulated environments and real world scenarios, enabling Floo to directly serve the real applications (§4.2). Extensive experiments demonstrate that Floo reduces the RCT by about 14.3% to 52.7% on average compared to using a static CCA. Further evaluation also shows that Floo is able to achieve satisfactory performance in the real world in different scenarios (§5).

In summary, our key contributions in this paper are:

- By demonstrating the difficulty of optimizing QoE for CCAs, we reveal the need for a practical QoE-oriented CCA selection mechanism (§2).
- We propose Floo, a QoE-oriented mechanism for CCA selection, which supports seamless switching on the fly for large-scale deployment of mobile web services (§3, §4).
- We deploy and evaluate Floo with Dianping service of Meituan. Our extensive experiments showed that Floo achieves consistent high performance under dynamic mobile networks (§5).

2 Motivation and Challenge

In this section, we use real-world mobile web service traces to demonstrate optimizing QoE for CCAs is extremely challenging in §2.1. Then, we present our design choices in §2.2 to address the mismatch between QoS and QoE. Finally, we elaborate on the challenges of designing the QoE-oriented CCA selection mechanism in §2.3.

2.1 Optimizing QoE for CCAs is extremely challenging

Convoluted relationship between QoS and QoE. QoE metrics are defined by applications. In contrast, QoS metrics focus on the descriptions of transport layer performance. The optimization of QoS is not consistent with that of QoE. For example, for large bulk requests, a reduction in RTT does not imply a QoE improvement [37]. Small request-intensive web pages are not that sensitive to throughput increase, since their total bandwidth need may still be small. As for the applications that apply recovery techniques such as FEC [32], packet loss also does not have a significant impact on QoE [25]. CCA has no idea what goal the application optimizes towards and whether application layer techniques are used. Therefore, the relationship between QoE and QoS is convoluted.

We conduct emulated experiments to demonstrate the convoluted relationship. Four well-known CCAs are considered: Cubic [23], BBR [12], Copa [9], and Westwood [14]. We use real-world traces extracted from Dianping to generate request-response messages (detailed in §4.2) on Mahimahi [41] emulated network paths. The WSP algorithm [44] is employed to compute the configurations of 100 different network path conditions (detailed in §4.3.1). Each CCA runs on each network path condition for 2 minutes. We calculate the metrics of Thpt, RTT, Power [27], etc., periodically at the sender according to transport layer acknowledgments. The results are presented

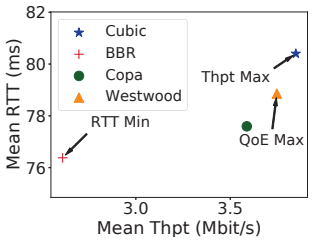


Figure 1: The mean throughput and RTT of the four CCAs achieved on a network path.

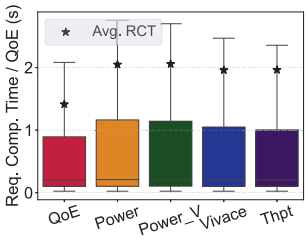


Figure 2: The RCT for all network path conditions following different CCA selection metrics.

Metrics	Definition
QoE	*Request completion time*
Power [27]	$Power = \frac{Throughput}{Delay}$
Power_V [28]	$Power\_V = \frac{Throughput*(1-LossRate)}{Delay}$
Vivace [19]	$Vivace = rate^t - b*rate*\frac{dRTT}{dT} - c*loss*rate$
Thpt	*Delivery rate*

Table 1: A QoE metric and four QoS metrics. QoE is calculated as the average RCT after the CCA convergence (20s after connection establishment). Power is the most common metric used for transport layer evaluation [7, 22, 31, 57]. Power_Variant [28] is one variant of Power, which also considers the packet loss rate. Vivace is used by an online learning CCA, and it is in form of utility function [19, 20].

in Fig. 1 and 2.

Fig. 1 shows the performance of the CCAs on a specific network path. Among the four CCAs, the CCA that achieves the highest throughput or the lowest RTT does not reach the best QoE (defined in Tab. 1). Then we consider the relationship between QoS and QoE on multiple network paths. For each network path condition, we select the best CCA following different metrics listed in Tab. 1. For each metric, we obtain all the RCT of the requests running the selected CCA for all network path conditions. Fig. 2 shows the Tukey boxplot for each metric. Results show that if the CCA is selected following the QoE metric, the RCT can be reduced by at least 27% on average, and none of the popular QoS metrics can achieve similar RCT to the QoE metric. This well demonstrate the convoluted relationship between QoS and QoE.

Mismatched timescale between QoE and QoS. The mismatch also exists in the time scale of the CCA and application optimization. CCAs collect fine-grained ACK information and make decisions at a granularity of packet-level or RTT-level. In contrast, QoE is measured and evaluated at a coarse granularity of the request level, usually around hundreds of milliseconds or even longer [9, 54]. As a result, it is difficult to map high time scale QoE to low time scale CCA behavior.

2.2 Design Choices

Instead of optimizing QoE for CCAs, we decide to design a shim, which selects an appropriate CCA aiming for better QoE. The QoE-oriented CCA selection approach addresses the mismatch between QoS and QoE:

- **Optimizing the real goal.** It is hard to make QoS-oriented CCAs optimize the QoE, because of the mismatch between QoE and QoS. However, with QoE metrics as the basis, the CCA selection shim allows the transport layer behavior to be optimized toward application layer objectives. As discussed in §2.1, the QoE is hard to be replaced by existing transport capability-oriented QoS metrics. Therefore, QoE is regarded as the real goal of our approach.
- **Time scale.** The CCA selection approach works above the transport layer to make decisions at a coarser granularity, understanding the QoE, and deciding which CCA to use. Specific sending rate/CWND increase or decrease decisions of the CCAs' are not necessarily closely coupled with QoE. Thus, the time scale mismatch is solved. The CCAs do not need to interact with the application layer and do not need to be modified.

2.3 Challenges

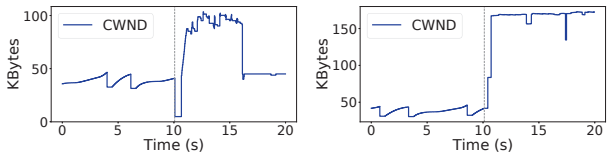
However, designing and implementing a CCA selection mechanism is non-trivial in a large-scale real-world deployment of mobile web service.

CCA Selection. Creating a mapping from the observed network conditions and QoE metrics to CCAs, or in other words, generating a CCA selection policy is challenging.

- **Fluctuating network conditions.** Under mobile networks, network conditions fluctuate due to wireless channel fading, user movement, or network congestion. Adapting to the dynamic network condition is challenging.
- **Empirical CCA characteristics.** The existing knowledge of the applicable scenarios of CCA is usually empirical [11, 15]. It is very difficult to model and characterize the CCAs, especially the recent proposed CCAs [9, 19]. Adapting to the complicated CCAs is challenging.

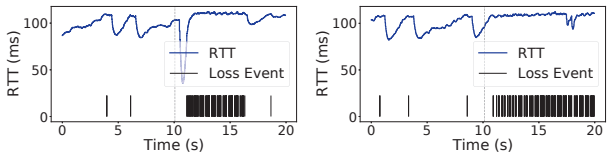
Smooth switching on the fly. Due to dynamic network conditions, CCA switching may occur during transmission. We consider two kinds of switching: Part Switching and Full Switching. While Full Switching makes the new CCA inherit all CCA-related variables, including connection-level variables (e.g., CWND/sending rate and RTT-related values), and CCA private state variables (e.g., fulled_pipe in BBR), Part Switching only inherits connection-level variables, and the private state variables of the new CCA are initialized from default values. To demonstrate their differences, we build a small testbed including two hosts and one switcher. The two hosts establish a QUIC connection, which continues to send massive data. One CCA switch event happens at the 10th second. As shown in Fig. 3, the switching without CCA state migration (Part Switching) has two problems:

- **Longer convergence time and performance deterioration.** Without CCA state migration, the new CCA starts at the slow start phase and the CWND or sending rate increases exponentially from the steady state of the previous CCA until it converges again. The path condition information required for the new CCA still needs time to be col-



(a) Cubic switches to BBR.

(b) Cubic switches to Copa.



(c) Cubic switches to BBR.

(d) Cubic switches to Copa.

Figure 3: Two demonstrations of Part Switching. The switch event happens at the 10th second. (a) and (b) show the CWND change over time. (c) and (d) show the smooth RTT changes over time and when loss events happen. Part Switching from Cubic to BBR leads to longer convergence time and performance deterioration. Part Switching from Cubic to Copa leads to distorted path estimation collected and results in abnormal behavior of the new CCA.

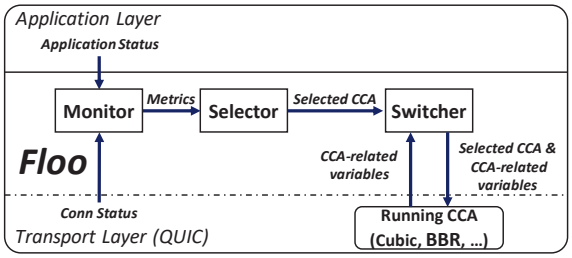


Figure 4: Floo Overview

Characteristic	Metric	Description
Btlbw	$Thpt\_max$	The maximum delivery rate
RTProp	RTT_min	The minimum RTT
Random Loss	$Loss\_rate$	|Lost Packet| / |Sent Packet|
Buffer	RTT_rate	Smooth RTT / Min RTT
RTT Variety	RTT_{var}	The variance of RTT samples
CCA	CCA_name	Current CCA
	$CWND/SendRate$	CWND or pacing rate
App-level	$Goodput$	The average Size / Duration of past responses
	$Unsent\_size$	Total bytes of the response waiting for writing to Send Buffer
	Qws	The average duration the response wait for writing to Send Buffer
	$Qws_{gradient}$	The gradient of the Qws samples
	$Bytes\_interval$	Bytes sent in the last SP

Table 2: The metrics collected by Monitor module.

lected and estimated. Worse still, since the previous CCA has converged basically, the new CCA's re-convergence will cause a lot of packets lost (about 14.8% within 2s after switching in Fig. 3c).

- **Distorted path estimation results in abnormal behavior of new CCAs.** The network condition observed by the new CCA reflects the condition after the previous CCA converged. It may not be consistent with the real path condition. For example, as shown in Fig. 3d, there is already a queue buildup in the buffer by Cubic when the switching occurs. Therefore, the *minRTT* observed by Copa is biased after the switching. In such a case, Copa is unable to make proper decisions to reduce the RTT or drain the queue. Fig. 3d shows that the high RTT may last for tens of seconds or even longer, which completely conflicts with Copa's design goal of low latency.

Therefore, smooth switching is necessary. Ideally, the new CCA should inherit all the CCA-related variables and continue to update them according to the newly observed network conditions after the switching. However, considering the more complex state design of emerging CCAs, and the personalized state variables, mapping the state of a certain CCA to a new one is much more challenging.

3 Design

3.1 Design Overview

Fig. 4 shows the overall architecture of Floo. We build the main building blocks of Floo atop QUIC, including Monitor module, Selector module and Switcher module. Monitor module collects information from both the transport layer and application layer. According to the state variables saving application statistics and connection statistics, Monitor module computes the metrics of interest in consecutive time intervals and feeds them to Selector module every selection period (SP) (§3.2). According to the metrics fed by the Monitor module, Selector module selects a CCA based on a pre-trained selection policy, which maps the metrics from Monitor module to appropriate CCAs. With the help of that policy, an appropriate CCA is selected to maximize the application QoE. The selection policy is pre-trained offline with RL methods (§3.3). After getting informed of the new optimal CCA, Switcher module conducts CCA state migration to complete a smooth switching from the current CCA to the new CCA (§3.4). Thus, an accurate and smooth switching of CCA is completed.

For a QoE-oriented CCA selection mechanism, one key design decision of Floo is how to incorporate QoE metrics into the mechanism. From emulated experiments in §2.1, we observe that it is difficult to replace QoE with QoS. As Fig. 2 shows, the CCAs selected with the QoE criterion achieve the lowest RCT, while the other four QoS metrics, Power, Power_V, Vivace and Thpt, have 31.01%, 31.34%, 27.87%, 28.01% higher RCT on average respectively. Therefore, our answer is to directly set QoE as the selection criterion.

3.2 Monitor

Monitor module gathers collectible statistics that can be measured and monitored, including the transport layer and application layer statistics. The transport layer statistics are collected from the information provided by ACK packets or the connection maintained state variables. As for the application layer statistics, we collect them from mobile web applications.

We consider the metrics shown in Tab. 2. The CCA selection mechanism is to select the optimal CCA for different network conditions. Therefore, we first consider the characteristics describing the network path conditions. They are

the former 5 rows of the table. We use bottleneck link bandwidth ($BtlBw$), round-trip propagation time ($RTProp$), RTT variety (RTT_{var}), packet loss rate ($RandomLoss$) and buffer size ($BufferSize$) to describe a network path. We estimate these objective network path conditions with the following collectible metrics: the maximum delivery rate ($Thpt_{max}$), minimum RTT (RTT_{min}), average loss rate ($Loss\_rate$), relative value of buffer size v.s. BDP (RTT_rate) and the RTT variety (RTT_{var}). The calculation of the metrics occurs in a slicing time window of 10s.

Note that the above metrics are only the observed metrics of the network paths and not the objective network path conditions. The relationship between the observed and objective values can be affected by the current running CCA. In order to accurately reflect the objective values of the network path, two additional types of information are collected. On the one hand, we record the current CCA, and $CWND$/$SendRate$ if applicable. On the other hand, we collect the information from the application layer, specifically the $Goodput$, $Unsent\_size$, Qws, $Qws_{gradient}$, $Bytes\_interval$ (defined in Tab. 2). It is worth noting that Qws is the response waiting time, which is the time between the response is generated on the server and the response data is written to the send buffer of the transport layer. Qws reflects the growth and drain of the queue in the send buffer. When the network condition worsens, the queue in the send buffer will pile up rapidly, making the rise in Qws. The calculation of these app-level statistics occurs every SP.

3.3 RL-based Selector

Selector module selects the optimal CCA based on the metrics passed by Monitor module. The selection policy, or the mapping from the Monitor module metrics to the optimal CCA, is pre-trained and saved in Selector module. We utilize an RL approach to build a prediction model as the selection policy, because RL and CCAs are similar, i.e., both of them continuously make decisions according to the changes of environment. In this section, we describe the RL system. The process and method of offline training are shown in §4.3.

State & Action. We use the metrics passed by Monitor module as the state of RL system, which is used to select the optimal CCA. We use normalized metrics instead of the exact values. This avoids exaggerating the impact of an input metric with very large values on the final model. Further, normalization helps Selector module generalize the network conditions it observes during the training phase to unseen network conditions and achieve better performance. As for the action, the RL system uses CCA candidates as the possible action values.

Reward. Reward is an important factor affecting the RL system's performance. Specifically, the rewards the RL agent gains at each step quantify its performance to improve its subsequent action. Numerous RL-based congestion control-related solutions adopt Power or its variants as reward [7, 28, 31, 42, 57]. However, our experiment results show that application-layer metrics are more appropriate reward indices than QoS-oriented transport layer metrics (§2.1). Therefore, we directly utilize the gradient of QoE as the reward for our RL system.

We regard the RCT as the QoE of web services, which is also a common QoE choice for such services. RCT is the time taken from sending the request to receiving the last byte of the response, recorded on the client side. RCT mainly includes the transmission elapsed time within the network (transmission time) and the response queuing time on the sender (i.e., Qws in Tab. 2). For web services, we employ this value as the RL system reward for the following reasons:

- When the sending rate is high, the response queuing time is much smaller than the transmission time. Therefore, RCT is approximately equal to the transmission time, which reflects the current transmission efficiency, and thus can evaluate the current action (i.e., CCA).
- When the sending rate is low, or is lower than the delivery rate from the sender application layer to the sender transport layer, RCT mainly includes the response queuing time. In this case, RCT reflects the growth and drain of the queue within the send buffer. If the current action is better than the previous action, it will suppress the queue growth or accelerate the queue draining. The change of the queue will be reflected in the change of RCT, and will further evaluate the merit of actions in one training round.

Specifically, we use the gradient of RCT as the reward: $R = \ln \frac{Last\ RCT_{avg}}{Current\ RCT_{avg}}$. In each step, we record the average RCT in time units and compare it with that of the last step. After one entire training episode, we normalize all the rewards uniformly.

Learning algorithm. Floo adopts actor-critic RL, and is trained using the Proximal Policy Optimization (PPO) algorithm [45]. PPO is an advanced RL algorithm that is adept at exploring policies with continuous features. PPO addresses the issues of the traditional policy gradient philosophy and improves the utilization of data and the model stability by the design of importance sampling and clipping. Appendix A details how PPO is utilized in Floo.

3.4 CCA State Migration

To deal with the challenges described in §2.3, our design goals are: (1) Inherit the network path estimation to speed up CCA convergence and avoid performance degradation. (2) Retain the characteristics of new CCAs consistent with the original design goals. In our design, the CCA state migration mechanism considers all the CCA-related variables, i.e., the CCAs' phase and the variables used in CCAs. This mechanism can be applied to non-learning CCAs, e.g. Cubic, BBR and Copa.

CCA Phase Migration. CCA phase migration is concerned with the state transition within the CCAs. With packet loss no longer the only congestion signal, the emerging CCA phases become more complex. However, a common feature of non-learning CCAs is that they all probe the path and estimate

Variable Type	Description
Sending rate variables	Variables that directly determine the sending rate, e.g. CWND, Pacing rate, etc.
Observation variables	Observations of the connection, e.g. smooth RTT, max delivery rate, etc.
Parameter variables	Variables related to CCA design, e.g. β=0.8 in Cubic when packet loss.
Other variables	Variables that maintain CCA's current state, e.g. *fulled_pipe* in BBR, and *velocity* in Copa.

Table 3: Four types of CCA-related variables.

their occupancy of the path based on the feedback. Therefore, we coarsely classify CCA phases into two categories based on how well the CCA probes the path.

The first category is the non-converged phase, i.e., where CCA has not formed a complete awareness of the path or does not fully utilize the available capacity. The non-converged phase includes both the slow start phase and the situation where the CCA would not fill the pipe for other purposes, such as ProbeRTT in BBR. The second category is the converged phase, i.e., where CCA adjusts the sending behavior based on the observations of the path after the slow start, including the congestion avoidance phase of traditional CCAs, the ProbeBW phase of BBR, and the moving phase of Copa.

We adopt different measures to migrate different phases. If the switching happens at the converged phase of the old CCA, Floo makes the new CCA directly enter the converged phase.This avoids massive packet loss caused by the slow start phase after switching. We do not perform switching at the non-converged phase. On the one hand, the statistics collected during the non-converged phase are unreliable. On the other hand, the non-converged phase usually does not last too long, so it will not cause much damage even if the switch is not made immediately. Note that there is one exception: if the new CCA is BBR, Floo makes the BBR enter the ProbeRTT phase first. Though the ProbeRTT phase is not converged, it will affect the performance of the converged phase.

CCA Variable Migration. CCA variable migration is mapping variables from the prior CCA to the new CCA. The variables maintained by various CCAs are different and affect CCA switching performance differently. Therefore, we group all variables into four types, as shown in Tab. 3. According to our two design goals, we adopt the corresponding migration methods for each type.

- **Sending rate variables.** Sending rate variables, such as CWND and pacing rate, directly determine the sending rate. Therefore, they need to inherit the prior rate, thus ensuring smooth switching. The key issue here is the conversion between the rate-based CCAs and window-based CCAs. We use the relationship that $CWND = pacing\ rate * RTT$ to calculate the migrated values.
- **Observation variables.** Observation variables are estimated statistics for the network path. The observation variables collected in the converged phase can directly follow the new CCA. Considering that some important observation variables are not preserved by all CCAs, we additionally preserve the bottleneck bandwidth and minimum RTT at the granularity of the connection.
- **Parameter variables.** Parameter variables are related to the design of the CCA, which are basically fixed values and will determine the performance. Therefore, we do not perform any manipulation on these variables.
- **Other variables.** Other variables maintain CCA's current states, most of which are computed from observation variables. We migrate them based on the phase migration methods. The left ones are simply initialized to default values.

4 Implementation and Training

We implement Floo atop QUIC in the production environment of Dianping service with O(10M) daily active users. We first introduce the implementation of Floo (§4.1). Then, in order to make our model applicable to real applications, we conduct a large scale passive measurement on Dianping application from Meituan, analyze the traffic patterns of the application and use the collected application traces for training (§4.2). Also, we train the RL-based CCA selection model with the numerous newly collected real application traces and wireless network traces (§4.3) to make Floo Selector module suitable for real application scenarios.

4.1 Implementation

We implement Floo based on QUIC [29][2] in user space. Floo only requires modification on the sender side. For the training phase, we implement Floo's RL-Agent on top of Tensorflow [6]. After the training phase, we obtain the trained model, which is used in Floo's Selector module. The training phase is well presented in §4.3.

As for the applications, we slightly modify Dianping to support Monitor module of Floo, then the modified Dianping can run on top of Floo. We use it in real-world experiments (§5.3). Additionally, we also implement a simple request-response messaging application (Application S) atop Floo, which is used in the training phase of Floo and all the emulated experiments in §5. Application S generates requests and responses according to the application traces introduced in §4.3.1. In the training phase of Floo, Application S negotiates with RL agent about the information of state, action, reward, etc. For all the emulated experiments in §5, Application S employs the well-trained model. The sender of Application S selects CCA dynamically according to Floo's Selector module.

CCA Candidates. We consider the CCAs that have been deployed in real Internet environments as candidates for our CCA selection policy. Firstly, for deployability, we mainly consider widely-deployed CCAs, and thus choose the most two widely-deployed [36] CCAs, Cubic and BBR. Secondly, for effectiveness, CCA candidates should cover diverse QoS metrics. Therefore, we also use the loss-resilient Westwood

[2]We use an IETF QUIC implementation, ngtcp2 [2].

and latency-sensitive Copa. These four CCAs have different preferences for QoS targets (§2.1).

4.2 Application Dataset

We measure the traffic patterns of a real mobile application in production environments. These measurements illuminate the nature of request-response messaging traffic and provide the basis for constructing CCA selection policies that can be used for real applications.

We perform a large-scale passive measurement on Dianping from Meituan. Users can make purchases through this application, and the main user actions include searching, viewing images, etc. When the users are using the application, the client establishes a persistent connection with a frontend server, through which the application sends requests to the frontend server. We instrument the mobile APP client of that application, and after each request is completed, the instrumentation collects application-level logs and connection-level logs describing the finished transport process. We collected the logs of about 35 million request-response messages over two weeks.

First, we found that the connections of the application are persistent and would last 206s on average, which is much longer than the SP, supporting Floo's CCA selection. Then, we present the characteristics of the mobile web service, i.e. size and frequency of requests and responses sent through the persistent connections between the client APP and the frontend servers in Fig. 5. Fig. 5a shows the CDF of the request size. As we can see, over 80% of the requests are less than 10 KB, indicating that most of the upstream traffic is small and generally not the performance bottleneck. Fig. 5b shows the CDF of the response size. The responses have a diverse mix of small and large sizes with heavy-tailed characteristics. For the response workload, more than 70% of the responses are less than 10KB, but more than 60% of all bytes are in the 3.4% of responses.

Fig. 5c shows the time interval between the two consecutive sending of requests from the client. The inter-sending time between requests reflects the density and diversity of requests initiated by the application. Since this interval is influenced by both user behavior and application characteristics, we filter out the request initiation due to the user behavior. Specifically, the two requests with an inter-sending time greater than 1 second are considered two clicks of the user behavior. After filtering, as shown in Fig. 5c, 80% of the request inter-sending intervals are less than 44ms, and 38.4% of them are concurrent (0ms). *Therefore, although most of the requests and responses are tens to hundreds of kilobytes in size, the bandwidth needs of the application are still high.*

4.3 Training

Floo's training goal is to learn one policy that can select an appropriate CCA to achieve good QoE in diverse network environments. That policy should be applicable to real applications over real network environments. For this purpose, we use real application traces and real wireless traces for training in a controlled emulated environment (§4.3.1). Further, we use real QUIC implementations and Application S, instead of network simulators (§4.3.2). This allows the RL agent to have an experience close to that in real-world scenarios.

Parameter	Value Range (Min - Max)
RTT(ms)	10 - 50, 50 - 100, 100 - 150, 150 - 300
RTT Jitter / RTT	0 - 0.2, Jitter max = 20ms
Loss rate(%)	0 - 0, 0 - 0.1, 0.1 - 5
Buffer / BDP	0.3 - 0.9, 0.9 - 1.1, 1.1 - 1.5

Table 4: Network condition parameters.

4.3.1 Trace and Training Settings

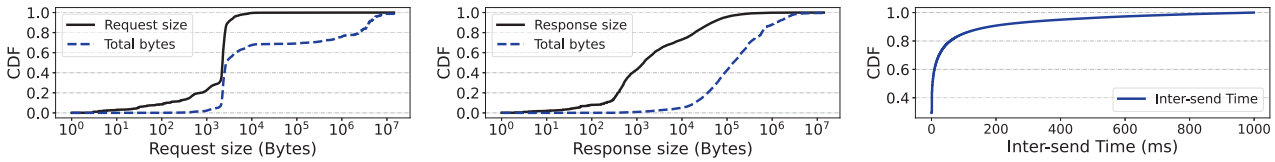
Application traces. We generate training application traces based on the distribution of the statistics collected from the measurements(§4.2). Specifically, we generate each request and response based on the CDF of the request and response size (Fig. 5a and 5b). The sending time of each request is determined based on the CDF of the inter-sending intervals (Fig. 5c). The server of Application S generates a corresponding response and delivers it down to the transport layer immediately after receiving a request. We train the RL model with many episodes, and each episode lasts 10 minutes. We generate a separate application trace for each training episode.

Network condition parameters. We use Mahimahi to emulate network paths and Traffic Control (TC) [8] to emulate RTT jitter. We adopt the network traces collected and used in previous works [3, 7, 30, 34, 38, 39, 47, 50], as listed in Tab. 6. These traces are employed to emulate the time-varying network path rate upper limit. They can be used to emulate various network conditions including 4G and 5G in both stationary and mobile scenarios.

Besides rate upper limit, RTT, RTT jitter, packet loss rate, and buffer size are also common network condition parameters [17, 52]. We select the values of these parameters using the space-filling WSP algorithm [17, 44] over the ranges listed in Tab. 4. WSP algorithm could generate multiple sets of network conditions based on the range of each parameter in order to emulate network conditions as diverse as possible. We generate 20,000 sets of network condition parameters. At the beginning of each training episode, we randomly select a network trace with one set of network condition parameters.

4.3.2 Training Method

We construct a training architecture consisting of learning agents, Application S clients and Application S servers. The client and server connecting to the same agent also establish a QUIC connection through Mahimahi. We set the episode to 10 minutes, which is long enough for CCAs to converge. For each training episode, the agent selects and configures the network condition parameters of Mahimahi and the application traces to be applied, as described in §4.3.1. The detailed training method is depicted in Appendix C.



(a) Distribution of the request size. **(b)** Distribution of the response size. **(c)** Distribution of the inter-sending interval.

Figure 5: Distributions of request and response characteristics from the measurements depicted in §4.2.

5 Evaluation

We first introduce our experimental setup in §5.1. We then evaluate Floo in the following aspects:

- **Consistent high performance.** We evaluate Floo over different scenarios. Evaluation shows that Floo achieves the highest throughput and lowest delay under different network conditions. Floo can reduce the application RCT by up to 52.7% on average, and up to 78.16% at the tail (§5.2).
- **Performance in the real world.** We implement Floo in a popular mobile web service and measure the performance for 96 hours. Experiments with real users show that Floo can reduce RCT by about 14.26% in the real world (§5.3).
- **Overhead.** Floo has acceptable overhead, with about 1.4% additional CPU utilization and sub-ms magnitude of additional time consumption (§5.4).
- **Improvement deep dive.** Finally, we evaluate the effectiveness of the design of Floo (§5.5).

5.1 Setup

We evaluate Floo in both emulated networks (§5.2, §5.4, §5.5) and large scale production environment (§5.3).

Emulated environment. We evaluate Floo in a controlled environment by emulating different network conditions with Mahimahi and generating new application traces. In our testbed evaluation, we implement Application S atop Floo, which sends requests and responses with application traces, and collects statistics for evaluation. We conduct experiments under 60 scenarios, including 10 stationary WiFi traces, 20 stationary cellular traces, and 30 mobile cellular traces. We also use the WSP algorithm to select 60 sets of other parameters, using the same method as §4.3.1, and importantly, the combinations of trace and parameter sets are different from those traces used in training. We compare the performance of Floo respectively with Cubic, BBR, Copa, Westwood and Vivace [19]. In each scenario, we send requests and receive responses using different algorithms for 3 minutes with the newly generated application traces.

Large scale production environment. We implement Floo in Dianping, with O(10M) daily active users. Our experiments are conducted in production environment where clients are heterogeneous including different OS, HTTP versions, etc. We manually enable Floo for a fraction of users, measure the performance for four days and collect 35 million request logs. In the experiments, we set the selection period (SP) as 12s. A further analysis of the impact of different SP values is detailed in Appendix F.

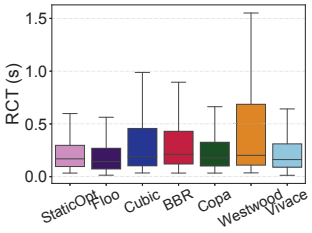


Figure 6: Floo achieves lower RCT than QoS-oriented CCAs.

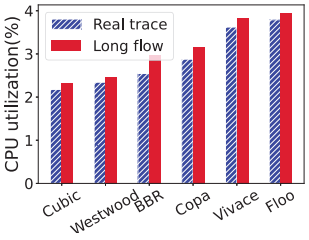


Figure 7: CPU utilization.

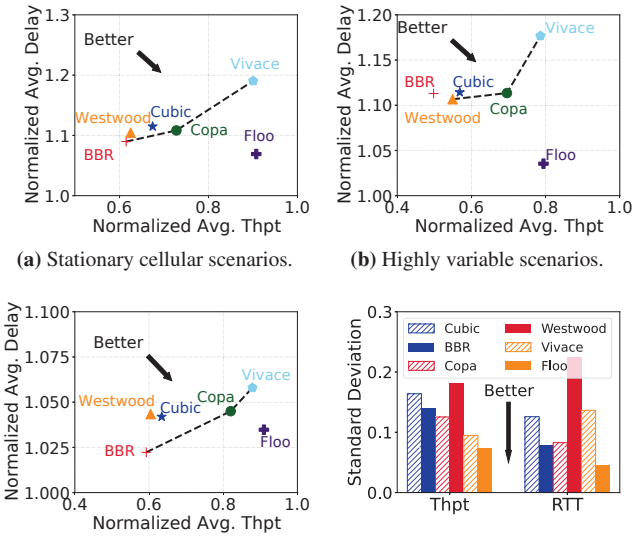
5.2 Consistent high performance

Here, we demonstrate that Floo achieves consistent high performance over different network scenarios.

Overall performance. We evaluate Floo under different scenarios, with 60 real-world traces. We record the RCT and show the performance of all scenarios in Fig. 6. For each scenario, we compare the four CCA candidates, and select the optimal CCA which achieves the lowest average RCT. The aggregated best choices for all scenarios is presented as Static_Opt. Floo achieves the lowest RCT, and reduces the overall RCT by a median of 20.11% to 32.54% and 21.18% to 78.16% in the 90^{th} percentile. The average RCT was reduced by 14.3% to 52.7% compared with QoS-oriented CCAs.

Remark 1 (Cubic and Westwood): Floo reduces the average RCT by 52.7% compared to Cubic and 50.8% to Westwood. For the 90 percentile (the tail) RCT, Floo shows great improvement, and has a 74.6% reduction compared to Cubic and 78.18% to Westwood. This is because that different types of CCA have different scopes of application. Empirically speaking, the performance of loss-based CCAs (i.e. Cubic and Westwood) degrades with high RTT and random packet loss, and will suffer longer tail latency. The improvement of Floo in the tail RCT demonstrates its selection accuracy to not use Cubic/Westwood when the network condition is poor.

Remark 2 (Copa and BBR): Compared to Copa and BBR, Floo reduces the average RCT by 20.53% and 14.3% respectively. For the 90 percentile (the tail) completion time, Floo has a 21.18% reduction compared to Copa and 21.57% to BBR. The improvement of Floo over the four CCA candidates validates the accuracy of our Selector module.



(a) Stationary cellular scenarios. **(b)** Highly variable scenarios.

(c) Wifi scenarios. **(d)** SD. of all scenarios.

Figure 8: A set of transport layer results for the three scenarios. (a)(b)(c) present the normalized avg. delay, and avg. throughput, where the dashed line represents the Pareto front of baselines. (d) presents the standard deviations of the overall results.

Remark 3 (Static_Opt): Floo can adapt to variable and dynamic network scenarios, and switch to a better CCA during the connection whenever the network condition changes. Therefore, Floo obtains better performance than statically optimal selection (9.6% reduction in avg. RCT). The advantage of Floo over Static_Opt demonstrates the need for CCA switching during the connection.

Remark 4 (Vivace): Vivace is also designed for performance. Therefore, we adopt Vivace as a baseline to evaluate Floo's ability to improve application QoE. Results show that Floo achieves 17.72% lower RCT on average than Vivace, and 25% reduction in the 90^{th} RCT. This is because Vivace still focuses on transport layer metrics and the utility function of Vivace is not consistent with the QoE. In addition, the penalty for packet loss and latency in the utility function makes Vivace less resistant to random packet loss.

Transport layer performance under different scenarios. In our emulated experiments, we consider three scenarios, including stationary cellular scenario, highly variable scenario and unseen WiFi scenario. We analyze the transport layer metrics under different scenarios. We consider two performance metrics: average smooth RTT and average throughput. For each scenario, we normalize the RTT and throughput performance of all CCAs (including Floo) to the minimum delay and maximum throughput achieved on that scenario, respectively. Then, we average all normalized values over all scenarios and show the results in Fig. 8. More detailed results are presented in Appendix E.

- **Stationary cellular scenarios.** In our evaluations, there are 20 traces of the stationary cellular network, including indoor [34] and outdoor [7] traces (Fig. 8a).
- **Highly variable scenarios - mobile cellular.** Similarly, we tested 30 mobile cellular traces, which are highly variable scenarios (Fig. 8b). These traces are collected when walking and driving under 4G [7] and 5G mmWave [38]. We also adopt the 4G measurements on high-speed rails [30] to construct a scenario with violently fluctuating bandwidths.
- **Unseen scenarios - WiFi.** To evaluate the behavior of Floo in unseen scenarios (Fig. 8c), we use WiFi traces that have not been employed in the training, We used 10 WiFi traces from [35], including traces from office and a public WiFi provided by a crowded restaurant during dinner hours.

Results show that Floo generally achieves the highest throughput with the lowest latency under different scenarios. Even in unseen scenarios, Floo shows advantages, demonstrating Floo's generalization capability. The improvement of Floo demonstrates that, besides QoE improvements, directly optimizing the application QoE through CCA selection approach can further improve transport layer capabilities.

We also present the stability of throughput and RTT (i.e., the standard deviation of the normalized average throughput and RTT) of each CCA under all scenarios in Fig. 8d. Results show that Floo could almost achieve stable high performance in all three scenarios, especially in terms of latency.

5.3 Real-world performance

We implement Floo in Dianping with O(10M) daily active users. We manually enable Floo for a fraction of users. Specifically, we deployed Floo on the front-end server to serve the persistent connection between the front-end server and the client. We enabled Floo for 5% of the users for evaluation. Besides Floo, we also implement a Floo with Part Switching (P-Floo), enable P-Floo for another 5% users and evaluate the effectiveness of switching algorithms. As a comparison, we set up another 5% of the users to use Cubic, BBR, Copa, Westwood and Vivace respectively. We collect logs for 96 hours, resulting in more than 35 million request logs, covering users from more than 50 countries and regions. We collect RCT from client side and the results are shown in Fig. 9 and Fig. 10.

Floo is still able to reduce the RCT and obtain optimal performance in real-world scenarios. Floo achieves a QoE improvement of 8.07% to 14.26% in real scenarios, with a reduction of about 25.5% for tail RCT. The difference in RCT between the real scenario and the emulated evaluation mainly comes from the different distribution of network conditions. For emulated evaluation, we aim to cover various network conditions by selecting as diverse network environments as possible. In contrast, the network states in the real scenario are not uniformly distributed. We found that there are about 57.89% of scenarios are under better network conditions (i.e., packet loss rate is 0% and min RTT is less than 44ms). Therefore, unlike the significant advantage of Copa in the emulated

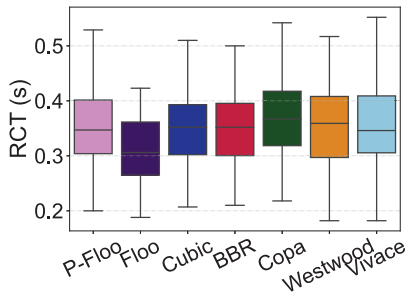


Figure 9: In real world experiments, Floo brought 12.9% reduction on average RCT.

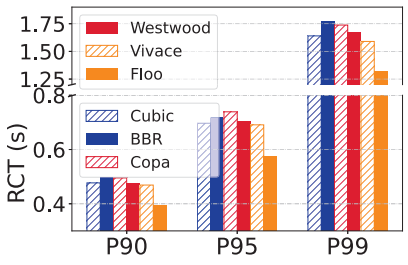


Figure 10: Floo reduces 25.5% of the 99^{th} percentile (the agestail) completion time.

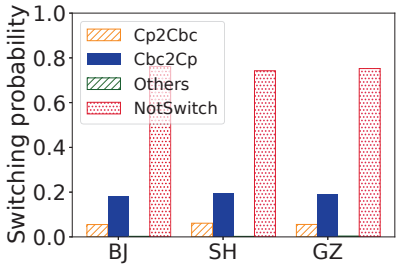


Figure 11: The probability of switching types of different user groups.

Part	CPU utilization (%)	Time Consumed (μs)
Network Monitor	0.784 (3.109)	47.4375
Selector	0.431 (3.540)	66.1875
Switcher	0.233 (3.773)	19.5625

Table 5: CPU overhead and time consuming of Floo.

scenario, Cubic and Westwood obtains a lower average RCT in the real world. As a result, the improvement brought by Floo in real world is a little lower.

We count the switching frequency and types in the real world in Fig. 11. With the SP of 12s, the probability of a CCA switching occurring at selecting is 25.76%, which implies an average switching interval of about 47s. Among the switching, the mutual switching between Copa (Cp) and Cubic (Cbc) is the most frequent, with more than 74% of the switching being Cubic to Copa and more than 23% being Copa to Cubic. We group users based on the location of CDN nodes they access. The results of switching frequency and actions in different groups are similar.

5.4 Overhead

We report the CPU utilization and runtime overhead of Floo.

CPU utilization. We measure the system overhead of Floo and compare it with other state-of-the-art CCAs. We performe experiments on an emulated network (with 48Mbps bottleneck link and 20ms RTT) for 6 minutes. We measure the average CPU utilization with real application traces and a long flow separately and show the results in Fig. 7. All algorithms are implemented atop QUIC in user space. Although Floo has a higher overhead compared to classical CCAs, however, compared to Cubic, which has the lowest overhead, the additional overhead is only 1.4%.

We measure the CPU utilization of each part by incremental experiments. Specifically, we separately measure the CPU utilization of only Monitor module, Monitor module with Selector module, and the complete Floo. We define the computed overhead of each part as the difference in CPU utilization (%) between two measurements. The results are shown in Tab. 5.

Time consuming. We show the time consumed by recording the time spent for each module. Tab. 5 presents the results taken as an average across 16 runs. We see that the additional consumed time introduced by Floo is at the sub-millisecond level, which is much less frequent than that of CCA selection (about 38.9s on average in our testbed experiments). Specifically, Selector module takes the most time (about 66μs) because of the complex calculations for CCA selection. Monitor module also consumes about 47μs to collect the additional information. Switcher module executes the state migration mechanism, which consumes about 19us.

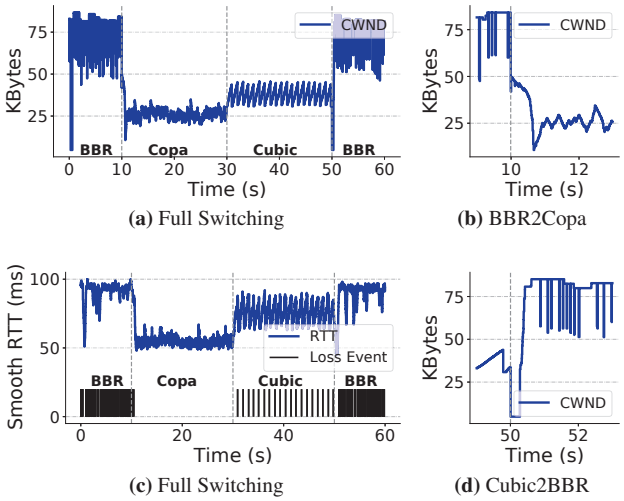
5.5 Floo deep dive

Here, we evaluate the effectiveness of Floo's design of state migration, generalizability to other QoE metrics and resilience to stochastic packet loss.

5.5.1 Effectiveness of state migration

Functional validation. Floo encounters situations where the path conditions change during transmission and switching-on-fly is required. To evaluate the effectiveness of Floo's state migration algorithm, we manually set the CCA switching every 20s and switch between all CCAs in an emulated environment. We compare Full Switching and Part Switching under 60 different scenarios.

Fig. 12 shows the details of the congestion control switching process. We show the performance of switching from BBR to Copa, and to Cubic. We do not show additional details of Westwood, since Westwood is basically similar to Cubic in terms of algorithm design. As described in §2.3, packet loss occurs under Part Switching when switching occurs without convergence. In addition, Copa maintains a high CWND and thus experiences a high RTT with packet loss due to the distortion of estimation of path conditions. Full Switching, as shown in Fig. 12, avoids these problems and maintains the CCA characteristics consistent with their design. Specifically, when switching to Copa (Fig. 12b), Floo is able to decrease the CWND within 1 second, thus quickly emptying the queue built by BBR and maintaining low latency. When switching to BBR (Fig. 12d), Floo first enters the ProbeRTT phase so as to obtain the RTProp, which the prior CCA did not maintain earlier. After that, BBR does not enter the Startup phase, but gradually converges with the ProbeBW phase.



(a) Full Switching **(b)** BBR2Copa

(c) Full Switching **(d)** Cubic2BBR

Figure 12: Details of Full Switching. (a)(c) Full Switching statistics of CWND, smooth RTT and loss event. (b) Detailed CWND of Full Switching from BBR to Copa. (d) Detailed CWND of Full Switching from Cubic to BBR.

Performance analysis. We compare the performance of Full Switching and Part Switching in both emulated and production environments. Fig. 13 shows the distribution of packet loss rate within 1 second after the switching in the emulated environment. Part Switching leads to more packet loss, while Full Switching significantly reduces the packet loss rate, especially when switching from Cubic to the low-latency algorithms, i.e. BBR and Copa. For real-world experiments, Fig. 9 shows that the average RCT is reduced by about 7.59% with Full Switching compared to Part Switching.

5.5.2 Generalizability to other QoE metrics

We change QoE to tail RCT in order to analyze Floo's ability to generalize to other QoE metrics. Accordingly, we set Reward as $R = \ln \frac{Last\ RCT_{p90}}{Current\ RCT_{p90}}$ and retrain a new RL model. We evaluate Floo-P90 under the same 60 scenarios as §5.2. For each scenario, we record the RCT of total requests, the RCT at the 50th percentile, and the RCT at the 90th percentile[3], respectively. We gather the value of all scenarios in Fig. 14.

Floo-P90 is not as good as Floo in terms of total performance of all requests. However, for the 90th percentile RCT, Floo-P90 has a significant improvement, with a reduction of 21.78% on average. Compared to Floo, Floo-P90 selects Cubic less frequently by 6.03%, while the frequency of using BBR, Copa and Westwood increased by 4.02%, 1.0% and 1.0%, respectively. This is because the loss caused by Cubic when filling the buffer can result in a long RCT, while BBR are relatively conservative. As a comparison, Floo reduces RCT at the 50th percentile by about 19.49% compared to Floo-P90. The above results show that with our mechanism, there can be a significant improvement on the target QoE metric. See §7 for more analysis on generalization ability.

[3]We do not compare the 90th percentile RCT of all secnarios because it represents the performance under poor network scenarios.

5.5.3 Resilience to stochastic packet loss

We also analyze Floo's resilience to stochastic packet loss. Stochastic packet loss often occurs under cellular networks due to channel interference, mobility, etc [24, 51]. We evaluate the performance of Floo with a single flow on a link with 4 Mbps bandwidth, 20 ms RTT, 10 KB buffer, and varying random loss rate from 0% to 10%. As shown in Fig. 15, Floo still maintains a low RCT when the stochastic loss rate is set to 10%. It is worth mentioning that Vivace maintains a low RCT until a loss rate of about 4%. After that, corresponding to the 5% loss resistance in the utility function [19], the average RCTs increase dramatically, even up to 9.5 times of the no packet loss case. In addition, the performance of Vivace suffers uncertainty and instability with random packet loss.

6 Related Work

QoE-oriented transport-layer optimization. There are many other ways to conduct QoE-oriented transport optimization [18, 19, 33], while conveying QoE to transport layer by CCA selection is more appropriate. QoS, the target of transport optimization, is reflected in the behaviors in the network, e.g. how to utilize the bottleneck queue, where CCA is the most effective procedure to control. For example, better packet scheduling could improve the host queueing time through reordering the packets [16]. However, packets from one application always have the same QoE, leaving little optimization space for packet scheduling. And flow control schemes could also decide the sending rate, while it is not aware of network behaviors. Therefore, conveying QoE through CCAs is more appropriate.

QoE-oriented CCAs. We are not the first to observe that CCAs should be optimized towards application QoE. One line of solution is to integrate application design and the transport layer behavior [13, 21]. However, these works are designed for specific applications and redesign is needed if designers want to migrate their good performance to other applications. There are also proposals to use application requirements, such as deadline [55] and priorities [56], to guide the design of CCA at the transport layer. However, they can only be used for application requirements that can be directly understood by the transport layer. For example, deadline can be identified as the data delivery time, which the transport layer can estimate and optimize directly from RTT and packet loss events. For the complex QoE metrics, a possible solution is to adopt mature algorithms, such as reinforcement learning, yet we found it impractical. If we put translated QoS as the goal of RL [7, 26], the gap between QoE and QoS remains. If we put QoE as the goal of the RL, the indirect and distant connection between QoE and cwnd/rate decisions makes the training extremely hard to converge. RL-based CCAs also

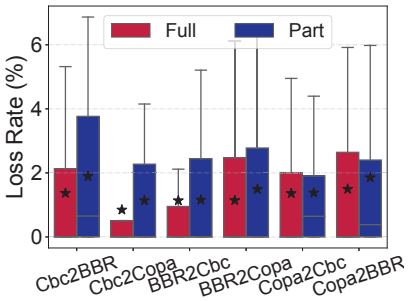


Figure 13: Loss rate of Full Switching and Part Switching within 1s after the switching. Cbc represents Cubic.

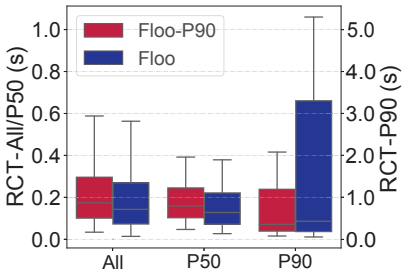


Figure 14: Tail-RCT-oriented Floo-P90 significantly reduces the 90th percentile RCT by 21.78% on average.

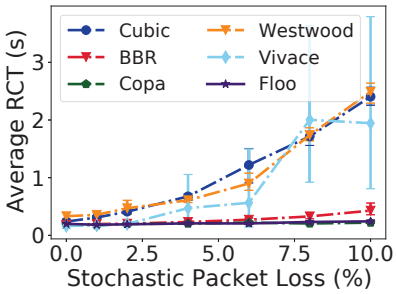


Figure 15: Impact of stochastic loss on RCT.

have interpretability issues in the wild. Therefore, Floo adopts CCA selection mechanism to bridge the gap between the complex, application-oriented QoE and the direct, transport capability-oriented QoS.

CCA-selection methods. There has been some work that adopts CCA selection methods [15,20,42,57]. Existing works still aim to optimize transport layer performance. However, as stated in §2.1, QoS-driven CCA-selection methods will suffer performance degradation when selection criteria are not entirely consistent with QoE. In addition, existing schemes cannot address the two challenges (§2.3) well. Their selection policy generation methods do not consider QoE, and they could not support seamless switching either.

Advanced CCAs for mobile web service. In recent years, many advanced CCA schemes have been proposed, including CCAs specifically designed for wireless network, such as Sprout [50], Verus [53], and emerging learning-based CCAs [7, 19]. However, these CCAs are still oriented to optimize the transport layer performance and do not address the QoS and QoE mismatch. Moreover, considering the practical issues [7, 15] and unproven performance in production environments, we did not consider them as CCA candidates.

7 Discussion

Generalization to heterogeneous applications and scalability to various CCAs. In this paper, we propose a solution for mobile web service, aiming to reduce RCT, and using four classical CCAs as candidates. In fact, Floo could be applied to heterogeneous applications and various CCAs. For example, for streaming applications, Floo can be reused by considering a chunk as a request. For complex QoE, as long as we could extract the traces, characteristics, and the QoE metric of the application, Floo can theoretically be used for any application without any idea of the implementation details and optimization techniques of the application. For CCAs, Floo can incorporate various CCAs into the selection mechanism. Nevertheless, CCA state migration has to be considered. Our design in §3.4 can be extended to all non-learning algorithms. However, generalizing to complex applications and switching between complex CCAs are not designed and verified in this paper, and are future work.

Fairness and friendliness. Floo selects among various CCAs, and the fairness and friendliness of Floo is consistent with that of the CCA candidates. In this paper, we select from the deployed CCAs, which already has had theoretical and experimental analysis of fairness and friendliness [9, 11, 48].

Portability to TCP in Linux kernel. Although Floo is implemented atop QUIC, Floo can still be applied to TCP implementation in Linux kernel. Firstly, eBPF technique [1, 10] provides a safe and convenient way to interact between user space and kernel. One can imagine that Floo works in user space, extracts information from the kernel and delivered the selected CCA to the kernel. The state migration mechanism, on the other hand, requires further modifications to the kernel. Secondly, as for the integration with mechanisms specific for TCP or QUIC, e.g., multi-streaming in QUIC, Floo is orthogonal to pre-CCA optimizations. Therefore, for implementing Floo over TCP without pre-CCA optimizations, Floo can also select the appropriate CCAs in respective situations.

8 Conclusion

We propose Floo, a QoE-oriented CCA selection mechanism for mobile web service. Floo uses QoE as the selection criterion and employs RL techniques to construct the mapping from the transport layer and application layer metrics to CCAs. Floo switches smoothly during the transmission. We implement Floo in a popular mobile web service, and evaluate Floo in both emulated and production environments. Experiments show that Floo reduces the RCT by 14.3% to 52.7% in different scenarios.

This work does not raise any ethical issues.

Acknowledgements

We sincerely thank our shepherd Wei Gao, anonymous reviewers, and labmates in Routing Group from Tsinghua University for their valuable feedback. The research was supported by the National Natural Science Foundation of China under Grant 62002192, Grant 62221003 and Meituan. Mingwei Xu and Enhuan Dong are the corresponding authors of the paper.

References

[1] ebpf. https://ebpf.io/what-is-ebpf/.

[2] ngtcp2. https://github.com/ngtcp2/ngtcp2.

[3] Raw data - measuring broadband america. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america.

[4] Desktop vs Mobile vs Tablet Market Share Worldwide. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet, 2022. Accessed: 2022-8-19.

[5] OpenAI-baseline. https://github.com/openai/baselines/tree/master/baselines/ppo2, 2022. Accessed: 2022-8-29.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[7] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 632–647, 2020.

[8] Werner Almesberger et al. Linux network traffic control—implementation overview, 1999.

[9] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *Proc. USENIX NSDI*, pages 329–342, 2018.

[10] Lawrence Brakmo. Tcp-bpf: Programmatically tuning tcp behavior through bpf.

[11] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. When to use and when not to use bbr: An empirical analysis and evaluation study. In *Proceedings of the Internet Measurement Conference*, pages 130–136, 2019.

[12] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[13] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

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

[15] Kefan Chen, Danfeng Shan, Xiaohui Luo, Tong Zhang, Yajun Yang, and Fengyuan Ren. One rein to rule them all: A framework for datacenter-to-user congestion control. In *4th Asia-Pacific Workshop on Networking*, pages 44–51, 2020.

[16] Yong Cui, Chuan Ma, Hang Shi, Kai Zheng, and Wei Wang. Deadline-aware Transport Protocol. Internet-Draft draft-shi-quic-dtp-06, Internet Engineering Task Force, July 2022. Work in Progress.

[17] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 160–166, 2017.

[18] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. Pcc: Re-architecting congestion control for consistent high performance. In *Proc. USENIX NSDI*, pages 395–408, 2015.

[19] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. Pcc vivace: Online-learning congestion control. In *Proc. USENIX NSDI*, pages 343–356, 2018.

[20] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. A unified congestion control framework for diverse application preferences and network conditions. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 282–296, 2021.

[21] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *Proc. USENIX NSDI*, pages 267–282, 2018.

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

[23] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[24] Habtegebreil Haile, Karl-Johan Grinnemo, Simone Ferlin, Per Hurtig, and Anna Brunstrom. End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks. *Computer Networks*, 186:107692, 2021.

[25] Yongkai Huo, Cornelius Hellge, Thomas Wiegand, and Lajos Hanzo. A tutorial and review on inter-layer fec coded layered video streaming. *IEEE Communications Surveys & Tutorials*, 2015.

[26] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

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

[28] Leonard Kleinrock. Internet congestion control using the power metric: Keep the pipe just full, but no fuller. *Ad hoc networks*, 80:142–157, 2018.

[29] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proc. ACM SIGCOMM*, pages 183–196, 2017.

[30] Li Li, Ke Xu, Tong Li, Kai Zheng, Chunyi Peng, Dan Wang, Xiangxiang Wang, Meng Shen, and Rashid Mijumbi. A measurement study on multi-path tcp with multiple cellular carriers on high speed rails. In *Proc. ACM SIGCOMM*, pages 161–175, 2018.

[31] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *Proc. ACM SIGCOMM*, pages 15–30, 2020.

[32] Michael Luby, Lorenzo Vicisano, Jim Gemmell, Luigi Rizzo, M Handley, and Jon Crowcroft. Forward error correction (fec) building block. Technical report, 2002.

[33] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. Pcc proteus: Scavenger transport and beyond. In *Proc. ACM SIGCOMM*, pages 615–631, 2020.

[34] Zili Meng, Yaning Guo, Yixin Shen, Jing Chen, Chao Zhou, Minhu Wang, Jia Zhang, Mingwei Xu, Chen Sun, and Hongxin Hu. Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning. *IEEE/ACM Transactions on Networking*, 29(2):723–736, 2021.

[35] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proc. ACM SIGCOMM*, pages 193–206, 2022.

[36] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The great internet tcp congestion control census. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–24, 2019.

[37] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning edge research with latency shears. In *ACM HotNets*, 2020.

[38] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand AK Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, et al. Lumos5g: Mapping and predicting commercial mmwave 5g throughput. In *Proceedings of the ACM Internet Measurement Conference*, pages 176–193, 2020.

[39] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, et al. A variegated look at 5g in the wild: performance, power, and qoe implications. In *Proc. ACM SIGCOMM*, pages 610–625, 2021.

[40] Ravi Netravali and James Mickens. Prophecy: Accelerating mobile page loads using final-state write logs. In *USENIX NSDI*, 2018.

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

[42] Xiaohui Nie, Youjian Zhao, Zhihan Li, Guo Chen, Kaixin Sui, Jiyang Zhang, Zijie Ye, and Dan Pei. Dynamic tcp initial windows and congestion control schemes through reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1231–1247, 2019.

[43] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V. Madhyastha. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proc. ACM SIGCOMM*, 2017.

[44] Jenny Santiago, Magalie Claeys-Bruno, and Michelle Sergent. Construction of space-filling designs using wsp algorithm for high dimensional spaces. *Chemometrics and Intelligent Laboratory Systems*, 113:26–31, 2012.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[47] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. Http/2-based adaptive streaming of hevc video over 4g/lte networks. *IEEE Commun. Letters*, pages 2177–2180, 2016.

[48] Ranysha Ware, Matthew K Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of the internet measurement conference*, pages 137–143, 2019.

[49] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. Http/2 prioritization and its impact on web performance. In *World Wide Web Conference*, 2018.

[50] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proc. USENIX NSDI*, pages 459–471, 2013.

[51] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. In *Proc. ACM SIGCOMM*, pages 479–494, 2020.

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

[53] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *Proc. ACM SIGCOMM*, pages 509–522, 2015.

[54] Jia Zhang, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Miao Zhang, and Yang Yue. Wisetrans: Adaptive transport protocol selection for mobile web service. In *Proceedings of the Web Conference 2021*, pages 284–294, 2021.

[55] Lei Zhang, Yong Cui, Junchen Pan, and Yong Jiang. Deadline-aware transmission control for real-time video streaming. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2021.

[56] Chao Zhou, Wenjun Wu, Dan Yang, Tianchi Huang, Liang Guo, and Bing Yu. Deadline and priority-aware congestion control for delay-sensitive multimedia streaming. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 4740–4744, 2021.

[57] Jianer Zhou, Xinyi Qiu, Zhenyu Li, Gareth Tyson, Qing Li, Jingpu Duan, and Yi Wang. Antelope: A framework for dynamic selection of congestion control algorithms. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

A Learning Algorithm of Floo

Floo's agent interacts with the environment, gets a series of trajectories ((state, action, reward) or (s, a, r) for short), and updates the policy according to the information of its interaction. Upon receiving state (s_t), Floo's agent needs to choose a corresponding action (a_t), i.e. one CCA, and will get the reward(r_t) in the next step. The policy that Floo's agent used to choose an action, is defined as the probability distribution of actions: $\pi(s_t, a_t) \rightarrow [0,1]$. $\pi(s_t, a_t)$ is the probability of taking action a_t in state s_t. Then the agent uses the PPO algorithm to update the parameter θ of the policy π. The PPO algorithm is optimized from the policy gradient methods [46], which estimates the gradient of the expected total reward by observing the trajectories obtained by following the policy. The gradient of Policy Gradient can be computed as:

$$\nabla_\theta J(\theta) = \mathrm{E}_{(s_t,a_t)\sim\pi_\theta}\left[\mathrm{A}^\theta(s_t,a_t)\nabla\log\pi_\theta(a_t^n \mid s_t^n)\right] \quad (1)$$

$\mathrm{A}^\theta(s_t,a_t)$ is the Advantage Function, which represents the difference in the expected total reward when we choose action a_t in state s_t, compared to the expected total reward for the action drawn from policy π_θ.

Policy Gradient is an on-policy method where the collected sample (s_t, a_t, r_t) is used only once. In order to make full use of the training data and improve the learning efficiency, PPO extends the Policy Gradient method. The original policy is denoted as π_θ, and when the gradient $\nabla_\theta J(\theta)$ is applied to the original policy π_θ, the new policy is denoted as $\pi_{\theta'}$. At this point, if we want to reuse the data generated by the policy π_θ to update $\pi_{\theta'}$, considering the different distributions of trajectories in $\pi_{\theta'}$ and π_θ, an importance sampling method is needed:

$$E_{x\sim p}[f(x)] = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right] \quad (2)$$

Therefore, the gradient of the off-policy is calculated as follows, with the parameters before and after the update denoted as θ and θ':

$$\nabla_\theta J(\theta) = \mathrm{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta'}(a_t \mid s_t)}\mathrm{A}^{\theta'}(s_t,a_t)\nabla\log\pi_\theta(a_t^n \mid s_t^n)\right] \quad (3)$$

Trace	Year	Type (Num)	Stationary / Mobile	Description
Lumos [38, 39]	2020	4G (166), 5G mmWave (121)	Stationary, Mobile (walking, driving)	Verizon's 4G and 5G service in Minneapolis.
NYC [7]	2019	4G LTE (23)	Stationary, Mobile (bus, taxi)	Cellular traces gathered in NYC.
PiTree [34]	2019	4G(61)	Stationary	Measurement of indoor 4G bandwidth.
HSR [30]	2018	4G (33)	Mobile (high-speed rails)	4G measurements on high-speed rails.
FCC18 [3]	2018	4G (397)	Stationary	The broadband network in 2018 provided by FCC.
Ghent [47]	2016	4G (40)	Mobile (foot, bicycle, bus, tram, train, car)	4G measurements in 2016 by Ghent University.

Table 6: The description of the real network traces.

The objective function is calculated as Eq. (4). To ensure that the difference between the policy before and after the update is not too large, PPO adds a constraint to the objective function. The clip function forces $\frac{\pi_{\theta(a_t|s_t)}}{\pi_{\theta'(a_t|s_t)}}$ between $1-\varepsilon$ and $1+\varepsilon$, and finally takes the minimum value among the rewards that have been clipped and those that have not been clipped.

$$J^{\theta'}(\theta) = \sum_{(s_t,a_t)} min(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta'}(a_t \mid s_t)} A^{\theta'}(s_t,a_t), \\ \text{clip}(\frac{\pi_{\theta(a_t|s_t)}}{\pi_{\theta'(a_t|s_t)}}, 1-\varepsilon, 1+\varepsilon) A^{\theta'}(s_t,a_t)) \tag{4}$$

The detailed derivation and sample code can be found in [5, 45].

B Real Network Traces

Tab. 6 shows the traces used in Section 4.3.1.

C Detailed Training Method

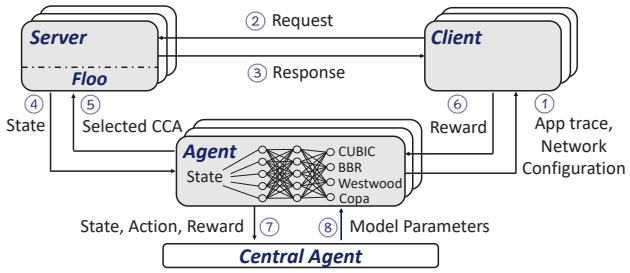


Figure 16: The workflow of the training phase of Floo. Multiple (Agent, Server, Client) sets run simultaneously.

We construct a training architecture as shown in Fig. 16. Each learning agent (Agent in Fig. 16) establishes two connections with an Application S client (Client in Fig. 16) and an Application S server (Server in Fig. 16) respectively, communicating about the experimental configurations and training trajectories. Each server/client only connects to one agent. The client and server connecting to the same agent also establish a QUIC connection through Mahimahi. For each training episode (10 min), the agent selects and configures the network condition parameters of Mahimahi and the application traces to be applied, as described in §4.3.1. During the interaction between the client and the server, the agent receives the states and rewards from the server and client respectively, and selects the corresponding action (i.e., CCA) based on the acquired state. The selected CCA is switched smoothly by Floo on the server.

To accelerate the training and improve the generalization performance of the RL model, we employ a distributed framework. We distribute 10 (Agent, Server, Client) sets. All the agents, servers, and clients are deployed in a cluster. These servers/clients are connected to the agents with high-speed links. Each agent observes a series of trajectories, and continuously sends the tuples (state, action, reward) to the central agent. The central agent then uses the PPO algorithm to compute the gradients (Eq. (3)) and updates the parameters in the selection model (Eq. (4)). The updated model will be pushed to each agent and will be used for the next episode. Tab. 7 in Appendix D shows the detailed model and parameters used during the training.

D Training Setting

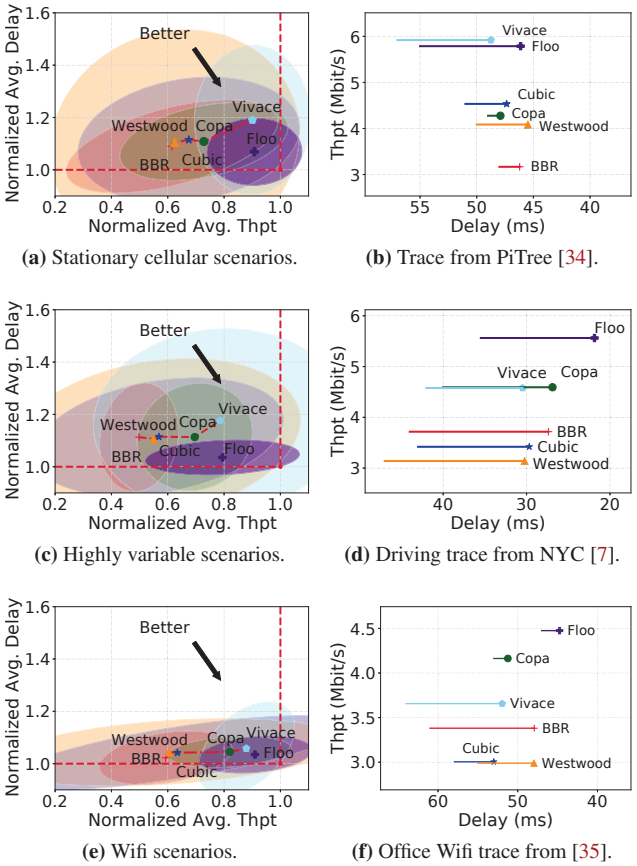
Floo uses an actor-critic architecture. Floo's actor, taking state and outputting action, use one hidden layer with 200 units. The output layer is a softmax layer to map to probabilities of actions. The critic networks, taking state and outputting V values, have one hidden layer with 200 units. The output layer is a linear unit representing the V function. All hidden layers in actor and critic networks are followed by Leaky ReLU nonlinearity. Tab. 7 shows other parameters used during the training of Floo.

Parameter	Value
Optimizer	Adam
Episode duration	10min
Actor's Learning Rate	0.0001
Critic's Learning Rate	0.0002
Discount Factor	0.99
ε in clip function	0.2

Table 7: Parameters used for the training.

E Detailed Results in Emulated Experiments

Our emulated experiments involve three scenarios: stationary cellular scenario, highly variable scenario and unseen WiFi scenario. We analyze the transport layer metrics, including average smooth RTT and average throughput. For each scenario, we normalize the RTT and throughput performance of all CCAs (including Floo) to the minimum delay and maximum throughput achieved on that scenario, respectively. We show the normalized results in Fig. 17a, Fig. 17c and Fig. 17e. The ellipse indicates the standard deviations from the average



(a) Stationary cellular scenarios. **(b)** Trace from PiTree [34].

(c) Highly variable scenarios. **(d)** Driving trace from NYC [7].

(e) Wifi scenarios. **(f)** Office Wifi trace from [35].

Figure 17: The detailed transport layer results for the three scenarios. (a)(c)(e) presents the normalized avg. delay, and avg. throughput. The center of each ellipse shows the average value of each CCA, while the ellipse indicates the standard deviations from the average values and their covariance. (b)(d)(f) shows the avg. delay (icons), 90% tile delay (end of lines), and avg. throughput of a sample.

values of the RTT and the throughput and their covariance[4]. Floo can not only achieve lower latency and higher throughput, but also obtain a smaller ellipse than other CCAs, which denotes better stability and consistency. In Fig. 17b, Fig. 17d and Fig. 17f we also depict the avg. delay (icons), 90% tail delay (end of lines), and avg. throughput of three typical sample traces. Under all three scenarios, Floo achieves excellent performance both in delay and throughput.

F Analysis of Selection Period.

Here, we investigate the impact of SP value on the performance and overhead of the mechanism. Intuitively, the SP determines the frequency of Floo monitoring network and application states, and selecting CCAs. SP should be consistent with the granularity of the application QoE and should also consider the network fluctuation. To this end, we vary the SP and record its impact on application performance. We

[4]Note that RTT/RTT_Min $\in [1,\infty)$, and Thp/Thp_Max $\in [0,1)$. The ellipse may exceed the range, but the outlier part is actually not sampled.

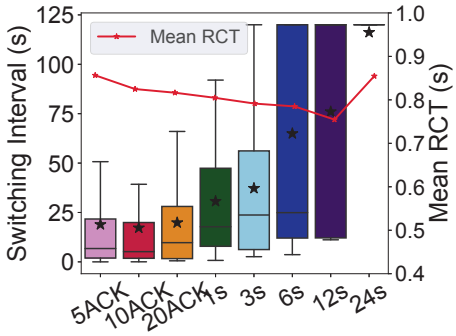


Figure 18: Avg. RCT (red line) and switching intervals distribution across diff. SPs

vary the SP from the fine-grained ACK level to the coarse-grained second-level. To report the performance, we conduct experiments in 60 scenarios and record the RCT and CCA switching interval. Note that if no switching has occurred in one scenario, the switching interval is counted as the duration of that experiment (120s).

Fig. 18 depicts the results. QoE gradually improves with the growth of the SP. The performance of ack-level selection is worse. We find that even for fine-grained ack-level SP, the granularity of CCA switching interval is at the second-level, with a median of about 5 to 10 seconds and an average value of more than 16 seconds. The QoE results are consistent with the frequency of CCA switching. This is because the fine-grained data estimation is susceptible to outliers, and cannot reflect the real path condition and application performance. On the other hand, the frequency of CCA selection is higher than that of request sending, which could lead to meaningless CCA switching. However, long SP, such as 24s, is challenged to capture and react to the instant changes in network conditions and application performance. In 96% of the scenarios, the SP of 24s does not switch during the connection. Therefore, long SP could not achieve good performance. As for the CPU utilization, experiments show that SP has little impact on the overhead. The difference in CPU utilization between different SPs is less than 0.28%. To have a balanced performance and overhead, we set a fixed SP of 12s in this paper, which represents the minimum switching period.