2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS) | 979-8-3503-9973-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/IWQOS57198.2023.10188765

A Data-Driven Framework for TCP to Achieve Flexible QoS Control in Mobile Data Networks

Jiaqi Yao*†§, Ke Liu†¶§, Ting Liang†¶, Theophilus A. Benson††, Jack Y. B. Lee‖, Vaneet Aggarwal‡‡, Yungang Bao†¶, Mingyu Chen†‡¶
*Henan Institute of Advanced Technology, Zhengzhou University
†State Key Lab of Processors, and Research Center for Advanced Computer Systems, Institute of Computing Technology, CAS
‡Zhongguancun Laboratory ¶University of Chinese Academy of Sciences ††Carnegie Mellon University
‖The Chinese University of Hong Kong ‡‡Purdue University

***Abstract*—Learning-based approaches have shown their great potential to adapt themselves to various environments (*e.g.*, PCC and Sprout). Unfortunately, they do not consistently achieve superior QoS across different network conditions and configurations in mobile networks. Furthermore, although they can offer multiple application objectives by adjusting a preference weight vector, it is challenging for users to accurately express an application objective with a weight vector. In this work, we argue that, if configured correctly, the delay-based TCP scheme can outperform learned ones, and allow users to directly specify their objectives. To this end, we propose Post-QoS Analysis (PQSA), a data-driven framework that trains the key QoS-impacting parameters of the scheme to capture the statistical correlations between QoS objectives, network conditions, and configurations, thereby determining the optimal parameter-set that meets the user-defined QoS objective under different network conditions and configurations. To support this, we enhance conventional delay-based TCP design to develop a Generalized TCP-like Rate controller (GR) by exporting three key parameters. Extensive evaluations show that PQSA-optimized GR outperforms existing schemes in different scenarios consistently, and enables service providers to control the QoS flexibly.**

I. Introduction

With the rapid advances in high-speed mobile networks such as 4G/LTE and 5G [8], various mobile apps with stringent Quality-of-Service (QoS) requirements are emerging, such as live video streaming, video conferencing, which requires real-time interaction with many users (*i.e.*, low delay), while maximizing the video/picture quality (*i.e.*, high throughput). Given that the mobile network does not provide any delay or throughput guarantee, the primary focus of the content provider or mobile operators nowadays is the development of rate control algorithms that achieves the application's desirable QoS – desirable delay-throughput tradeoff, by incorporating the inherent characteristics of mobile networks, *e.g.*, bandwidth fluctuations. To address it, a plethora of Congestion Control Algorithms (CCA) has been proposed, which can be classified into two design spaces:

TCP-like Scheme. Its CCA designs usually rest on certain assumptions about the network and application objective, and hard-wire predefined events to predetermined actions [1], [2], [7], [9]–[11], [21], [22], [24], [27], [29], *e.g.*, TCP-Westwood assumes the existence of non-congestion-related losses, thus avoids halving the Congestion Window (CWnd) blindly by taking into account the bandwidth at which congestion occurs [27]. However, when those assumptions are not held, performance degrades dramatically. In other words, conventional TCP-like CCAs cannot achieve consistent QoSs in different conditions without being manually tuned for each one [5].

Learning-based Scheme. Its CCA designs employ learning approaches (*e.g.*, Deep Reinforcement Learning (DRL)) to adapt themselves to various conditions [3], [5], [12]–[14], [18], [23], [26], [28], [30], [32]–[35], thus avoiding tuning or engineering for every single one of them. However, as will show later (§II), the learning-based CCAs, despite being trained under a wide range of network environments, can and do perform very inconsistently across *different network conditions (e.g., bandwidth fluctuation degree, propagation delay), network configurations (e.g., mobile base station buffer size), and application behaviors (e.g., flow duration)* presumably because their learned models are *not* specialized trained for each of them [5], [14], but optimized for a general one. As such, ultimately the resultant QoS is highly dictated by actual *network conditions or configurations*.

Moreover, the recent learning-based CCAs support multiple application (or QoS) objectives by incorporating a weight vector of throughput and delay into their training [14], [23], [26]. However, they leave the burden of weight vector configurations to users who have no clue to map the vector to application-level objectives. In specific, a weight vector is introduced to indicate the significance of the three primary performance metrics, namely throughput, latency, and packet loss rate [14], [23], [26]. However, converting the application objective into a specific weight vector for diverse applications or services, such as interactive live streaming like online games, which require a delay of tens of milliseconds, is a challenging task.

Therefore, both CCAs, despite being designed to function in mobile networks of all sizes and shapes, failed to achieve consistent QoS under different network conditions (*e.g.*, from 3G networks with a few *Mbps* peak bandwidth to 4G/5G networks with tens or hundreds of *Mbps* peak bandwidth; from

[§]Equal contribution

979-8-3503-9973-8/23/$31.00 ©2023 IEEE

TABLE I
Dataset Statistics. *The dataset #1 to #4 are collected by authors and the rest of datasets are publicly available. S stands for Stationary and M stands for Mobile.*

Location	Network Type	Service Provider	Trace Duration	Mobility	Throughput (Mbps)	Variation (CoV)
#1	4G	SMT	177h	S	15.04	0.63
#2	3G	SMT	169h	S	2.31	0.66
#3	3G	SMT	161h	S	4.87	0.32
#4	3G	SMT	168h	S	7.49	0.19
#5	4G	AT&T	2038s	M	5.20	0.89
#6	3G	TMobile	1859s	M	2.23	0.69
#7	4G	TMobile	948s	M	13.84	0.8
#8	4G	TMobile	279s	S	16.86	0.64
#9	3G	Verizon	2126s	M	0.66	0.37
#10	4G	Verizon	2728s	M	9.85	0.92
#11	4G	Verizon	280s	S	4.98	0.68

lightly loaded, well-covered mobile cells to crowded/congested cells), different network configurations (*e.g.*, from shallow buffered networks to deep buffered networks), and different application behaviors (*e.g.*, various flow durations). What's more, they do not allow users to directly and explicitly specify the QoS objectives (*e.g.*, in terms of delay) but an intricate weight vector.

Our insight is that the QoS of existing TCP-like CCAs can be improved, and even outperform learned ones if their internal parameters are tuned appropriately. This implies that it is possible for the parameters to capture the statistical correlation between the QoS objectives, network conditions, and configurations. Additionally, delay-based TCP-like CCAs are less sensitive to the network environment changes than learning-based ones (see §II-B), which makes specific training for every environment much easier. Thus, one question to ask is: *can we make the internal parameters configuration of TCP-like CCA network-aware, config-aware and QoS-objective-aware, and automatically apply the optimal configuration accordingly?*

To this end, we first design a Generalized TCP-like Rate controller (`GR`) that adopts the conventional delay-based CCA design but generalizes it by exporting three QoS-affecting key parameters. Second, as depicted in Figure 5, we propose a unified framework called Post Quality-of-Service Analysis (`PQSA`) with the design goal not only achieving consistent QoS, but also controlling the QoS flexibly. Specifically, `PQSA` allows users to specify application objectives directly in terms of maximum tolerable delay – *target delay*. `PQSA` then begins with the offline ensemble training phase that exploits simulated `GR` sessions under different *states* – the combinations of QoS objectives, network conditions and configurations, and determines the optimal parameter-set ensemble offline, where each one is optimized for a state, *i.e.*, meet the target delay while maximizing the throughput. The optimal parameter-set ensemble is then applied to the online ensemble adaptation phase where actual `GR` sessions are running. If the bandwidth trace used in training covers most of network conditions, the offline ensemble training phase does not need to be done repeatedly unless significant network upgrade (*e.g.*, 4G upgrade to 5G), and the optimal parameter-set ensemble is robust regardless of network operators (*e.g.*, SMT, AT&T), network types (*e.g.*, 3G, 4G), user mobility, *etc.*(see §V).

TABLE II
Bandwidth fluctuation degree for different throughput levels

Throughput Levels	0-1	2-3	4-5	6-7	8-9
Throughput Range ($Mbps$)	0-2	2-4	4-6	6-8	≥ 8
Coefficient of Variance (CoV)	0.84	0.59	0.42	0.32	0.25

Through extensive trace-driven emulations and real experiments conducted in production mobile data networks, we show that (a) `PQSA-GR` outperforms existing TCP-like CCAs and learning-based CCAs across a wide range of network conditions and configurations (*e.g.*, bandwidth fluctuation degree, propagation delay, buffer size, and flow duration); (b) `PQSA` enables users (*e.g.*, service/content provider) to explicitly specify the application objective via target delay, and incorporates the network conditions and configuration to automatically meet the target delay consistently, while maximizing the throughput.

II. Motivations

In this section, we study the unique properties of mobile networks, and then revisit the design and QoSs achieved by existing CCAs, both of which raise some fundamental questions on CCA designs and motivate our work.

A. Mobile Networks Properties

Observation 1: Mobile network conditions vary and fluctuate drastically based on a number of factors, such as, time-of-day, locations, mobility, service providers, etc. The end result of this fluctuation is that CCA must effectively adapt to these underlying network fluctuations. To illustrate, in Table I, we characterize and summarize many productions mobile networks. Dataset #1 to #4 were collected in 4 locations of downtown respectively, using a laptop connecting to a local network operator (SMT) [4], while the others are publicly available [9], [33]. We observe that despite using similar cellular technologies, the network characteristics can vary widely, *e.g.*, the mean bandwidth of SMT 3G in three locations (*i.e.*, #2, #3 and #4) varies by 224% at most.

Observation 2: The bandwidth fluctuates differently in different throughput levels. We divided dataset #4 into segments of $30s$, and classified every segment into one of 10 throughput levels if the segment's average bandwidth is within the throughput range of that level, where the throughput levels are shown in Table II. By grouping segments in levels, we can measure the bandwidth fluctuation degree of every level using Coefficient-of-variation (CoV). We observe from Table II, that bandwidth fluctuation degree differs across different levels: lower levels have larger bandwidth fluctuations, while higher levels have smaller fluctuations, *e.g.*, 0.84 versus 0.25, which is reasonable as a lower level means poor radio signal quality, bad coverage, *etc.*. Motivated, our solution adjusts the send rate with different aggressiveness for different levels, which is governed by protocol parameters.

Observation 3: Unlike traditional home networks and wireless networks where network queues are shared between different users in a mobile network, the base station maintains a separate large size queue for each user [21], [22], [33], thus *queueing delay can be a promising signal for congestion*

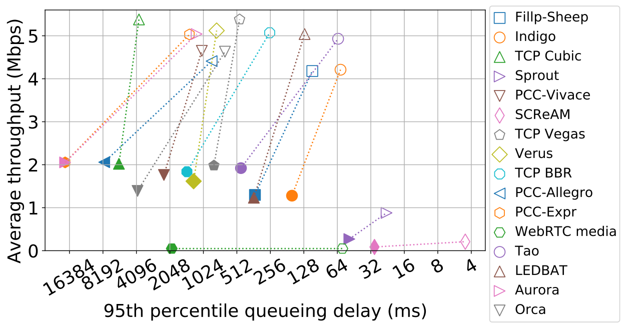

Fig. 1. **QoS of existing CCAs in different locations.** *We show that the QoS of existing CCAs can be very different in two different locations, where filled symbols denote the results of location #2 and nonfilled symbols denote results of location #3.*

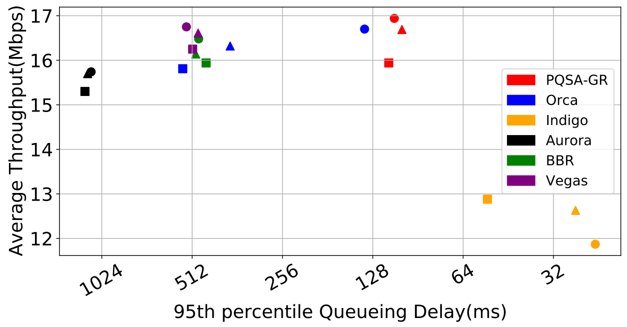

Fig. 2. **QoS of CCAs for different propagation delay.** *We show that* `PQSA-GR` *is not sensitive to the change of propagation delay. The circle represents the propagation delay of 20ms, the triangle represents 50ms, and the square represents 100ms. Each color represents a distinct learning-based CCA.*

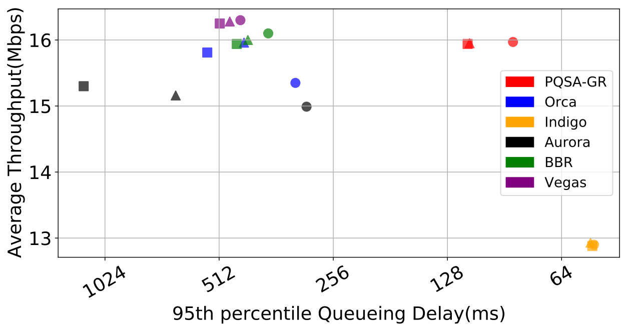

Fig. 3. **Qos of CCAs under different buffer sizes.** *We show that* `PQSA-GR` *is not sensitive to the change in buffer size. The circle, triangle, and square indicate the QoS results under the buffer size of 200 MTU, 500 MTU, and 1000 MTU, respectively, while each color represents a distinct learning-based CCA.*

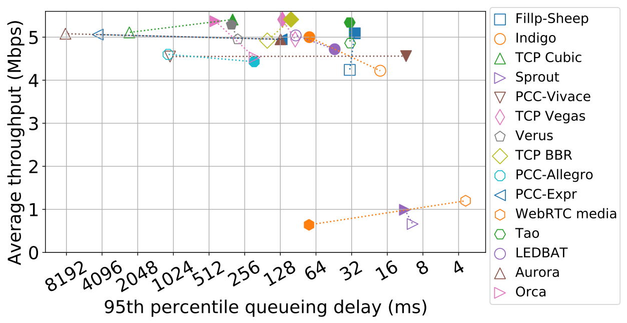

Fig. 4. **QoS of existing CCAs with different flow durations.** *We show that the QoS of existing CCAs can be very different with different flow durations. where filled symbols denote the results of flow duration 10s and nonfilled symbols denote the results of flow duration 120s.*

detection. Consequently, fair sharing of the radio link among multiple users is already enforced by the base station, thus fairness is not as crucial in mobile networks [22], [33].

B. QoS of Existing CCAs

We evaluated the QoSs of existing CCAs (TCP-like and learned ones) under different scenarios including network conditions (NC), Network conFigurations (NF), and Application Behaviors (AB). To evaluate them in a consistent and controlled way, we used trace-based emulation built with `Pantheon` [34], a platform for researching and evaluating end-to-end networked systems and transports, which replicates the bandwidth trace from dataset #1 to #4 in Table I to emulate a mobile radio link. We set the default propagation delay and emulated link buffer size of the testbed to be $100ms$, and 1K MTU, respectively. A sender sends a dummy data flow to a receiver via that mobile link. The default flow duration is 30s. The detailed experiment setup can be found in §V-A.

(NC#1) Location. We observe that a CCA's QoS does vary substantially across different locations, each with different bandwidth fluctuation degrees. Figure 1 shows the QoS comparisons across two locations, #2 and #3. The key observations are: (1) very few CCAs consistently maintain a small delay when moving between locations; (2) the learning-based CCAs do not necessarily outperform traditional TCP-like CCA, *e.g.*, LEDBAT achieves the best QoS in this case. This is presumably because learning-based CCAs are not trained specialized for every condition, thus experiencing serious performance issues over unseen conditions. Second, their models might converge slowly or converge to a wrong equilibrium [5], [23] (*e.g.*, PCC-Vivace [13] and PCC-Expr [3] underperform due to slow convergence (see more in §V)).

(NC#2) Propagation Delay. To study the impact of propagation delay on QoS, we used dataset #1 and evaluated existing CCAs' QoSs under network propagation delay of $20ms$, $50ms$, and $100ms$, respectively. Figure 2 shows that, although learning-based CCAs incorporate those delays into their training (*e.g.*, Orca sets the propagation delay to be $4ms$ to $400ms$ in its training), their QoSs vary with different propagation delays, *e.g.*, the queueing delay can be increased by four-fold for the state-of-the-art learning-based CCA, Orca. In contrast, the delay-based CCA obtains the consistent QoS. This is because they keep estimating the propagation delay (*i.e.*, minimum RTT).

(NF#1) Buffer Size. Modern mobile networks commonly employ a large radio link buffer to absorb large bandwidth fluctuations [24], [33]. Thus, we also studied the impact of the mobile link buffer size on QoS. We used dataset #1 and evaluated existing CCAs' QoSs under the link buffer sizes of 200 MTU, 500 MTU, and 1K MTU. Figure 3 shows that learning-based CCAs' QoSs are also sensitive to the change in link buffer size, although they incorporate a large range of buffer sizes into their training (*e.g.*, 3KB to 96MB for Orca). The Delay-based CCA (*e.g.*, Vegas) is also robust to the buffer size change. This is because it always controls the amount of queueing data within a small range, regardless of buffer size.

(AB#1) Flow Duration. We also observe that flow duration, an often neglected factor in existing works, impacts the QoSs of CCAs. In Figure 4, we analyze a CCA's QoS with two flow durations, *i.e.*, $10s$ and $120s$, which are typical durations for short-video [37] and video on demand respectively [36]. We observe that CCA's QoS is often *not consistent* with different flow durations, presumably because longer flows mean more network condition variations need to adapt, and some CCAs (*e.g.*, Aurora, PCC-Expr, *etc.*) do not actively reduce the transmission rate to empty the bottleneck buffer, which leads to more and more packets queued in the buffer when the flow duration is longer, significantly impairing the resultant QoS. This issue becomes particularly problematic for video streaming in mobile networks where the flows are often significantly large, and the networks vary quite rapidly.

C. *Multi-QoS objectives*

Recent multi-objective leaned CCAs incorporate different application objectives into model training [14], [23], [26]. Although users can express the application objective as a weight vector over multiple network-level metrics (throughput, delay, loss, *etc.*), users might have no clue to map weight values to their application objectives. A user-friendly system should allow users to specify the QoS objective explicitly and directly (*e.g.*, target delay $100ms$), instead of adding an extra layer to transform the QoS target into several weights.

D. *Takeaways*

The above results show that existing learning-based CCAs are very sensitive to network environment changes (in terms of network conditions, network configurations, and application behaviors). Although they are trained in a wide range of environments, they do not always perform better than TCP-like schemes which have fixed heuristics. The root cause, we conjecture, is that their training has incorporated a wide range of network conditions and configurations, thereby the resultant models are too general to be accurate/optimal for a specific operating environment (*e.g.*, Cov of bandwidth is 0.84, the propagation delay is 100ms, and the buffer size is 1K MTU).

An intuitive approach is to divide all potential network environments into states and train a unique model for each specific state. However, the state space can be large as it includes multiple dimensions from bandwidth fluctuation, propagation delay, buffer size, *etc.* The insight from the above results is that the delay-based CCA can outperform the learned ones if their internal protocol parameters are appropriately tuned for the current network state. Additionally, they are insensitive to buffer size and propagation delay changes (*e.g.*, BBR and

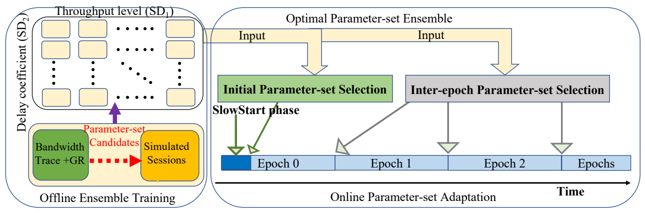
Throughput level (SD_1)
Delay coefficient (SD_2)
Parameter-set Candidates
Bandwidth Trace +GR
Simulated Sessions
Offline Ensemble Training
Optimal Parameter-set Ensemble
Input
Input
Initial Parameter-set Selection
Inter-epoch Parameter-set Selection
SlowStart phase
Epoch 0
Epoch 1
Epoch 2
Epochs
Time
Online Parameter-set Adaptation

Fig. 5. **`PQSA` system overview.** *`PQSA` contains two phases: offline ensemble training, and online ensemble adaptation.*

Vegas) and can be insensitive to flow duration (see §III). Thus, we only need to classify the bandwidth provision and fluctuation, which minimizes the state space.

In the end, we propose `PQSA` that classifies network bandwidth, Then, it takes a simulated approach, which simulates delay-based TCP CCA offline and searches optimal parameter-set satisfying the QoS objective for each class. This approach allows users to define their objective directly via *target delay*.

III. Post Quality-of-Service Analysis

As shown in Figure 5, `PQSA` runs on the server-side, and it consists of two phases. The first phase is offline ensemble training, where it employs a data-driven simulation approach to generate a complementary ensemble of tcp-governed parameter-set that are optimized for distinct two-dimensional states, defined by the target delay and network condition. Each parameter-set includes the bandwidth estimation window, queueing delay threshold, and rate adaptation interval, which are trained to meet a target delay while maximizing the throughput under a given network condition.

In the online ensemble adaptation phase, `PQSA` allows users to specify target delay according to the application's needs. By incorporating the target delay prescribed, `PQSA` re-uses delay-based TCP-like CCA (*i.e.*, `GR`) and leverages the information from the CCA to determine the network condition, in turn, selects the appropriate parameter-set to tune its control logic to adapt time-varying network conditions.

In this section, we first introduce our generalized view of delayed-based TCP (*i.e.*, `GR`) in §III-A and then discuss `PQSA`'s two-phase control loop in §III-B. Finally, we discuss its design choices in §III-C.

A. *`GR`: Generalized Delay-based TCP CCA*

We described the minimal requirements that `PQSA` makes of the TCP implementation that it enhances. At its core, `PQSA` retains the general delay-based CCA logic but only requires that the implementation expose a predefined set of parameters that significantly impact QoS. Note that re-using existing TCP implementation enables incremental deployability and ensures that the protocol works in all environments. while learning-based CCA breaks away from TCP control loop, this is difficult to implement in a kernel module (thus they were implemented in user-space or required to modify the kernel), and are not compatible with existing TCP sender and receiver implementation, which hinders their deployments [5].

To understand the parameters that impact QoS we first describe key aspects of TCP's CCA: (1) *time-window*, many delay-based TCP CCAs [10], [11], [22] calculate and estimate the network bandwidth based on the number of AckNum packets received during a predefined time window (*e.g.*, BBR uses a time window of six to ten RTTs to estimate the bandwidth). This window's size often enables the algorithm to make a trade-off between different timescales on bandwidth estimations: the longer the window, the more sensitive to longer timescale bandwidth variations, but less sensitive to short timescale bandwidth variations, and vice versa. (2) *Rate-adaptation-frequency*, in response to bandwidth estimates, the CCAs naturally adapt their sending rates. The frequency with which these rates are updated has a significant impact on the network. More frequent updates might lead to bursts and congestion, while less frequent updates might lead to insufficient bandwidth utilization. (3) *Delay-bound*, with delay-based TCP, *e.g.*, BBR [11] or Vegas [10], try to minimize the queueing delay. However, in practice, it cannot effectively balance the throughput and delay, and instead, result in bandwidth underutilization and significant rate oscillations.

In principle, PQSA interoperates with any delay-based TCP that can be modified to expose a parameter-set of these three parameters: bandwidth estimation window W, queueing delay threshold T, and rate adaptation interval Δ. We designed a delayed-based TCP CCA we can be augmented to expose the parameter-set easily, $\{W, \Delta, T\}$ denoted by $\mathbb{R}$ – General delayed-based Rate controller (GR). Given these three parameters, PQSA can effectively control the CCA behavior to operate in different network conditions effectively. Modifying existing delay-based TCP CCAs, *e.g.*, Vegas [10] and BBR [11], to expose the three parameters and evaluate their performances when cooperating with PQSA can be our future work.

Abstractly, a GR continuously measures the RTT when receiving an AckNum, Moreover, it estimates the bandwidth from AckNum returning rate within an estimation window W, which is denoted by B. At the end of every rate adaptation interval Δ, GR checks the measured RTT and tracks the minimum RTT measured, RTT_{min}: if $RTT > RTT_{min} + T$, GR sends at the rate less than B, *i.e.*, $\frac{RTT_{min}+T}{RTT}B$, which tries to decrease the queueing delay back to T. If $RTT < RTT_{min} + T/2$, GR sends at the rate higher than B, *i.e.*, $(1 + \frac{RTT_{min}+T/2-RTT}{RTT})B$, which improves bandwidth utilization by increasing the queueing delay to $T/2$. If $RTT_{min} + T/2 \leq RTT \leq RTT_{min} + T$, GR remains the sending rate intact, which becomes a safe cushion to minimize rate oscillations.

B. PQSA Control Loop

In Figure 5, we present PQSA's control loop.

Offline Ensemble Training. This phase simulates and analyzes GR sessions or flows with different parameter-set combinations over the past bandwidth trace. The goal is to determine an *optimal* parameter-set ensemble, each of which delivers the best QoS under a distinct state. PQSA tries to decompose the space of all potential network bandwidth levels and target delays into distinct states, each state has its own optimized parameter-set. This divide and conquer approach enables PQSA to consistently outperform other approaches but introduces several key challenges: First, how do we divide the space into a group of meaning states. Second, how do we quickly and effectively determine the parameters for each states. Third, as the network evolves, *e.g.*, due to the introduction of new technology (4G upgrade to 5G), their parameters need to evolve appropriately with new bandwidth traces.

Defining States: Our intuition for defining states builds on the key observations: (1) the bandwidth fluctuation is throughput-dependent (recall in §II), and (2) applications impose different target delays [26], thus we naturally view all states along two network dimensions (SD): throughput (SD_1) and delay coefficient (SD_2). We divide the throughput into N distinct levels and the target delay into M distinct classes.

Quantizing Throughput: One challenge for throughput quantization is that, while the throughput can be calculated as the bandwidth trace is given, it cannot be known at the beginning of an actual GR session. Therefore, we need a way to estimate the throughput level. Second, GR sessions may vary in length from seconds to minutes or hours (*e.g.*, live video streaming). Recall in § II-A, network conditions vary over time and a long session is unlikely to experience the same network conditions, thus it is unlikely for the entire session to have a single throughput level. We account them by subdividing trace into *epochs* of length P, and training GR independently for each epoch. Thus, each actual GR session can be viewed as a set of *epochs*. To estimate the throughput of a new epoch j, PQSA measures the throughput over the previous epoch, denoted by V_j (*i.e.*, the last γP second in epoch $j-1$):

$$V_j = \frac{\sum_{\forall i}\{a_i | t_j - \gamma P \leq r_{j-1,i} \leq t_j\}}{\gamma P} \tag{1}$$

where t_j is the beginning time of epoch j, a_i is the number of bytes acknowledged by TCP ACK i, $r_{j,i}$ is the arrival time of ACK i in epoch j, and γP is the time window estimating the throughput, where $\gamma \leq 1$.

PQSA uses a linear quantization policy (2) to map estimated throughput from epoch j to discrete throughput level L_j:

$$L_j = \min(\lfloor \frac{V_j}{\omega} \rfloor, N-1) \tag{2}$$

where ω is the quantization step size, and N is the number of throughput levels.

Applying it to all epochs simulated, the next step is to segregate trace of all epochs, *i.e.*, $S_j | \forall j$ into N classes:

$$C_n = \{S_j | L_j = \langle n \rangle, \forall j\}, n = 0, 1, ..., N-1 \tag{3}$$

where each class will emulate a particular throughput level for the training.

Quantizing Delay Coefficient: Unlike throughput, for target delay the metric of interest is often the 95th or the 99th percentile, *e.g.*, the 95^{th} percentile queueing delay $100ms$, which we characterize as ρ and Q, *i.e.*, the ρ^{th} percentile queueing delay Q. Training for large ρ or small Q will train parameters with shorter delay at the expense of throughput, and vice versa. `PQSA` employs M combinations of Q and ρ, denoted by $\{Q_m, \rho_m | m = 0, 1, ..., M-1\}$, to generate M target delay constraints for a epoch j , *i.e.*, let $q_{j,i}$ denote the i^{th} queueing delay sample estimated when `AckNum` i is received during epoch j, and $\{q_{j,i}\}$ denote the set of queueing delay samples:

$$D_m = \{Q_m, \rho_m | \frac{\sum_{\forall i | q_{j,i} <= Q_m} 1}{|\{q_{j,i}\}|} \geq \frac{\rho_m}{100}\}, m = 0, 1, ..., M-1 \tag{4}$$

which is used for evolving M distinct delay coefficient classes.

Consequently, we have a total of $M \times N$ states.

Performing Offline Training: For each state, we apply a separate training process, Divide, Diverge, and Simulated annealing (DDS) (see § IV), to obtain the optimized parameter-set, denoted by $\mathbb{R}_{n,m}$, which makes all the epochs in throughput level n, *i.e.*, C_n, satisfy the target delay constraint D_m constructed by one of M delay coefficients, while maximizing the average throughput of C_n, *i.e.*,

$$\max \frac{\sum_{\forall S_j \in C_n} b_{m,j}}{|C_n|}, s.t. D_m \tag{5}$$
$$n = 0, 1, ..., N-1, m = 0, 1, ..., .M-1$$

where $b_{m,j}$ is the resultant receiving throughput of S_j with $\mathtt{SD}_2 = m$. We provide details on DDS training in § IV.

Online Ensemble Adaptation. **Incorporating Application Preferences:** During the online phase, users optionally specify a target delay, $\lambda = \{\rho_{m^*}, Q_{m^*}\}$, class from M $\mathtt{SD}_2$ classes. Given the target delay, `PQSA` tries to maintain this target first, then maximize throughput second.

Performing Online Parameter-set Adaptation: During the online phase, `PQSA` initially lets `GR` use traditional TCP slow-start to bootstrap then switches to using the parameters learned during the offline training phase. For a new session, `PQSA` begins with TCP's default slow-start algorithm, hystart [16], and waits until exiting the slow start phase before it starts applying the optimized parameter-set. Recall, the goal of TCP's slow-start is to determine the current network capacity. To keep the throughput estimation method consistent with Equation 1, `PQSA` uses the network estimates (*i.e.*, throughput estimates, denoted by V_1) collected during in *epoch 0* – TCP slow-start phase, to determine the initial parameters:

$$V_1 = \frac{\sum_{\forall i} \{a_i | t_1 - \gamma P \leq r_i \leq t_1\}}{\gamma P} \tag{6}$$

where t_1 is the starting time of epoch 1, *i.e.*, the exciting time of the slow start phase of a `GR` session, V_1 is then used to estimate $\mathtt{SD}_1$ – throughput level for epoch 1 of an online `GR` session, using Equation 2.

Parameter-set Re-optimization: As discussed earlier, to compensate for long time scales bandwidth variations, an online `GR` session can be viewed as a set of epochs. Afterward, `PQSA` re-evaluates the parameters after the end of every epoch of P seconds. At the beginning of the next epoch, `PQSA` uses the knowledge from the last epoch to determine the session's current throughput level, and choose the appropriate parameter-set for the new network condition. To align with the offline training, `PQSA` uses the throughput estimates from Equation 1, V_j is then used to estimate the throughput level of epoch j, denoted by n^*. Thus, both m^* and n^* decide the optimal parameter-set $\mathbb{R}_{n^*,m^*}$ for epoch j.

Minimizing Re-optimization Frequency and Correcting For Estimation Error: `PQSA` tries to determine if there are potential errors by comparing the actual queueing delay to a predefined threshold. In case of no error, `PQSA` can re-use the previous epoch's parameter-set without parameter-set re-optimization – minimizing re-optimization frequency.

Specifically, if the actual queueing delay of epoch j, q_j, measured using $\mathtt{SD}_2 = m^*$, is larger than Q_{m^*} beyond a prescribed threshold ε, *e.g.*, 25%, *i.e.*, $(q_j - Q_{m^*})/\ Q_{m^*} > \varepsilon$, the network bandwidth available decreases substantially, which triggers `PQSA` to re-optimize the parameter-set by determining $\mathtt{SD}_1$ for epoch $j+1$ with Equation 1 and 2, and applies the corresponding parameter-set for epoch $j+1$ as shown in Parameter-set Re-optimization. If q_j is smaller than Q_{m^*} beyond a prescribed threshold ε, *i.e.*, $(Q_{m^*} - q_j)/\ Q_{m^*} < \varepsilon$, it implies the bandwidth is not utilized with the previous parameter-set, and cannot determine the $\mathtt{SD}_1$ correctly via throughput estimates (Equation 1) for epoch $j+1$, so `PQSA` simply increases the $\mathtt{SD}_1$ by one.

C. Discussions

We envision that `PQSA` is deployed at the servers, *e.g.*, content/service provider's servers [15] (*e.g.*, Tencent, TikTok), and our current reuse of TCP implementations ensures that `PQSA` is compatible with any client implementation. Next, we discuss the key design choices that ensure that it is practical for today's content provider deployments:

First, the two-phase design guarantees that, unlike most learning-based CCAs, `PQSA` offloads all learning complexity into the offline training phase. This choice minimizes aggregates and amortizes the resources used by the learning algorithm. During the online phase, each server needs only to perform simple calculations, *i.e.*, throughput calculation. Second, to improve accuracy, during the online phase, we reuse throughput estimates from slow-start instead of using the throughput of the previous session [6], [31]. Our approach eliminates potential errors that occur when sessions are separated by long periods (recall in § II, we show significant network variability over long periods). Last, `PQSA` is designed to interoperate with delay-based TCP CCAs, which allows `PQSA` to benefit from existing kernel optimizations, which other user-space ones are unable to do, *e.g.*, learning-based CCAs like Sprout, PCC.

Quic [20], a user-space transport framework on top of UDP that can implement any application-specific CCA. If `GR` or a delay-based TCP CCA is realized in Quic with the parameter-set exported, it also can be optimized by `PQSA`. As such, our work is orthogonal to Quic.

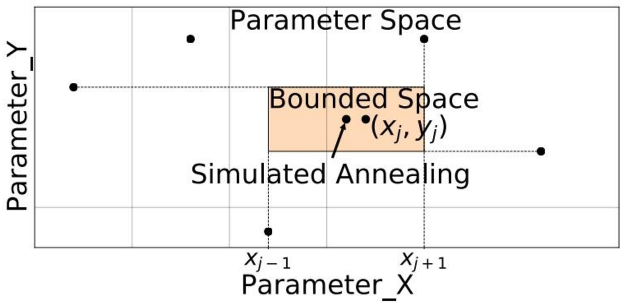

Fig. 6. **An example of running DDS in two-dimensional space.**

IV. Implementation

In this section, we present our experience with the implementation of `GR` and `PQSA` in practice.

`GR` kernel module: We implemented `GR` in a Linux kernel in the same manner as the existing TCP CCA, which leverages the pluggable TCP congestion control module handler interfaces, except that we expose the three parameters outside with `proc` interfaces, where `PQSA` process can access and overwrite parameters. To support online parameter-set adaptation, we leverage the existing `cong_avoid` interface that monitors the timestamps of received ACKs, and checks whether the elapsed time from the last parameter-set adaptation exceeds the epoch period.

`PQSA` trace-driven simulation: Mobile network bandwidth can vary widely in practice (see § II) and so far no known existing model can accurately capture their characteristics [24], [25]. Therefore, we primarily employ trace-driven simulations as a mean to evaluate `GR` (or different CCAs) in a realistic, consistent, and repeatable manner, where the mobile link varies based on bandwidth trace, and simulates `GR` sessions. We validated the fidelity of the simulator by showing that the simulated results are consistent with the emulated ones.

Offline ensemble training: To find $\mathbb{R}_{n,m}$ for a given state, where $n = 0, 1, .., N-1, m = 0, 1, ...M-1$, `PQSA` needs to simulate `GR` sessions with all possible parameter-set candidates from the three-dimensional parameter space, *i.e.*, $\{W, \Delta, T\}$, – *brute-force approach*. Although it is good for accuracy, its time complexity can be significant. To achieve an efficient tradeoff between accuracy and time complexity, we propose *Divide, Diverge and Simulated annealing (DDS)* to address this challenge: to reduce the parameter sample space, after dividing every parameter into k_0 intervals, DDS does not make a full combination of all samples, but takes a permutation of intervals for every parameter. DDS then aligns the permutation for each parameter dimension, to obtain a sample set $\mathbb{K}_0$ with size k_0. From $\mathbb{K}_0$, DDS finds parameter-set $\mathbb{R}_0$ that satisfies target delay, while maximizing the throughput. DDS then searches a *bounded space* around $\mathbb{R}_0$ using simulated annealing, as there is a higher possibility that DDS will find a "better" point $\mathbb{R}_1$ around $\mathbb{R}_0$, *e.g.*, point (x_j, y_j) in Figure 6. This bounded space, *i.e.*, the one bounded by the lower and upper initial value of parameters needed for simulated annealing, should not include any other samples in $\mathbb{K}_0$ except for $\mathbb{R}_0$. As shown in Figure 6, for parameter x in $\mathbb{R}_0$ we find x_{j-1} and x_{j+1} along the parameter dimension x of $\mathbb{K}_0$, where x_{j-1} is the largest point in $\mathbb{K}_0$ but smaller than that of $\mathbb{R}_0$, and x_{j+1} is the smallest point in $\mathbb{K}_0$ but larger than that of $\mathbb{R}_0$. Thus, (x_{j-1}, x_{j+1}) bounds the parameter dimension x. Above also applies to the other parameters.

Computation overhead: We showed the brute-force approach has the highest accuracy in finding the global optimal parameter-set at the expense of extremely high time complexity, *i.e.*, 4845 hours for training with dataset #1, while DDS has 6.1 hours for training the same dataset, without sacrificing the accuracy significantly.

TABLE III
`PQSA` System Parameters.

Notations	System Parameters	Default Values
N	$\texttt{SD}_1$	40
M	$\texttt{SD}_2$	3
γP	throughput estimation window	$250ms$
ω	Quantization step size	$1Mbps$
ρ, Q	Delay coefficients	$\rho = 95$ $Q = 50, 100, 200ms$
ε	Threshold to parameter-set re-optimization	0.25
P	Epoch period	$30s$

V. Performance Evaluation

Besides running simulations for the offline training phase, in this section, we leverage emulated experiments for the online adaptation phase to validate the performance of `PQSA` and compare against cutting edge learning and TCP-like CCAs.

A. Experiment Setup

For our emulations, we used a local trace-driven emulator based on `Pantheon` codebase [34], which allows us to dynamically load different CCA implementations and explore behavior under emulated network bandwidth captured in predefined datasets. We used 11 datasets that were captured from different mobile operators and scenarios as summarized in Table I, which efficiently covers most potential mobile network bandwidth. We note that these datasets were used to evaluate the prior works like [9], [22], [33], [34]. Datasets #1 to #4 were collected in a stationary position; thus, we repeated the experiments using datasets from #5 to #11, part of which were collected during driving.

`PQSA` Training: We only feed 33% of datasets #1 to #4 into `PQSA`'s offline training phase. Then, we used 67% of datasets #1 to #4, and all the datasets #5 to #11 from other operators for the online adaptation phase. This is used to demonstrate the *robustness* of `PQSA` regardless of network operators, network types, *etc.*.

`PQSA` system parameters: `PQSA` includes several parameters, we empirically analyzed several parameters and settled on the best-performing ones. For epoch size, we use $30s$, and evaluate its sensitivity in § V-D. To quantize throughput dimension($\texttt{SD}_1$), we used $N = 40$ with quantization step size of $1Mbps$. For delay coefficients, $\texttt{SD}_2$, we adopted $M = 3$ with $\rho = 0.95$ and $Q = 100, 150, 200ms$. We note that these parameters can vary by the content operators based on offline

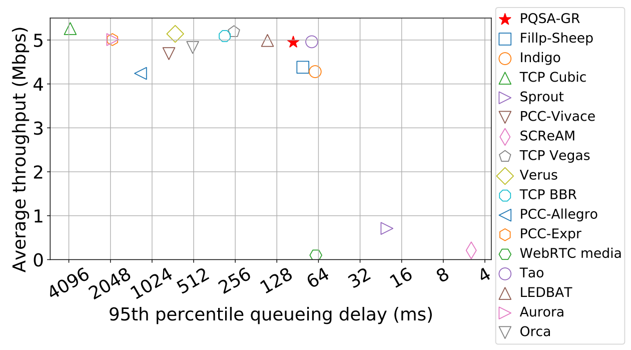

Fig. 7. **QoS of CCAs in 3G/HSPA.** *We tested all CCAs with unseen datasets from #2 to #4, and showed that* `PQSA-GR` *achieves the best QoS in 3G/HSPA.*

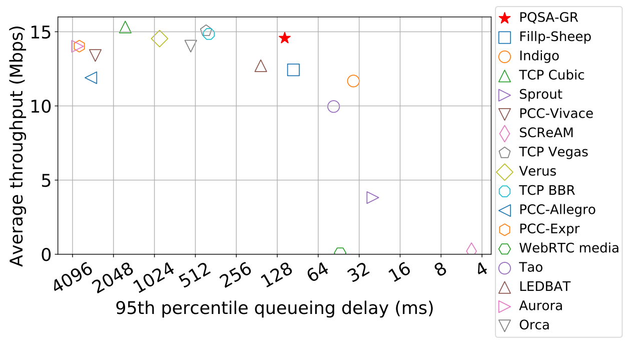

Fig. 8. **QoS of CCAs in 4G/LTE.** *We tested* `PQSA-GR` *with unseen dataset #1 and showed that* `PQSA-GR` *achieves the best QoS in 4G/LTE,*

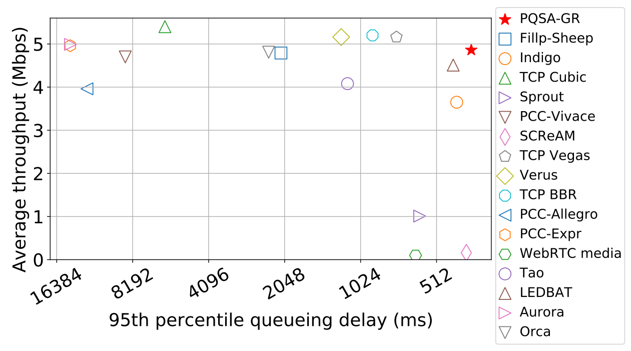

Fig. 9. **QoS of CCAs in mobile scenarios.** *We tested all schemes with mobile traces from datasets from #5 to #11 and showed* `PQSA-GR` *achieves the best QoS in mobile scenarios.*

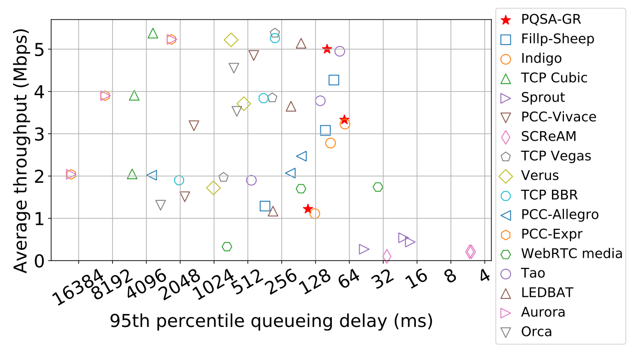

Fig. 10. **QoS of CCAs in three locations.** *We showed that* `PQSA-GR` *is spatial robust as it achieves the consistent QoS in different locations.*

analysis of the collected traces and application objectives. The rest of the parameters are summarized in Table III. Unless specified, we set `PQSA-GR`'s target delay to 95th percentile queueing delay $100ms$ (*i.e.*, $\rho = 0.95, Q = 100ms$), which is the benchmarks used in prior works [9], [22], [33]–[35].

Experiment summary: We evaluated `PQSA` from the following directions: (1) We demonstrate the QoS performance gain of `PQSA-GR` compared to prior CCAs [2], [3], [5], [7], [10]–[13], [17]–[19], [29], [30], [33]–[35] under different scenarios with emulated experiments; (2) We validate `PQSA-GR`'s robustness and flexible target delay control, and compare it to prior works. (3) We deep-dive into `PQSA-GR` design from different aspects, such as rate adaptation, parameter/algorithm sensitivity, and fairness.

B. *Overall Performance Comparisons*

In Figures 7 and 8, we compared the overall QoS performance of `PQSA-GR` to prior CCAs in 3G/HSPA+ (trace data from #2 to #4) and 4G/LTE (dataset #1) networks, respectively. We present the 95th percentile packet queueing delay against the average throughput of different CCAs.

We observe that `PQSA-GR` consistently outperforms all CCAs across mobile network types (from 3G to 4G), while the competing CCA's performance varies across network types illustrating that they do not perform consistently. Unsurprisingly, TCP-like CCAs do not necessarily perform worse than learned ones, *e.g.*, BBR versus PCC and Tao [30]. Moreover, learned CCAs, *e.g.*, Sprout, achieve low delay by sacrificing throughput. `PQSA` provides both, and provides predictable performance by delivering the actual 95th percentile queueing delay close to the target delay $100ms$. This is because `PQSA` creates a unique parameter-set that overfits each state.

Figure 9 plots the QoS results obtained by running emulated experiments with dataset #5 to #11 (from different network operators), which contain traces with mobility. `PQSA-GR` also performs consistently, and outperforms the other CCAs in mobile scenarios. As `PQSA` uses dataset #1 to #4 for the offline training phase, and applies the trained parameter-set ensemble in online testing with different datasets from other operators, `PQSA` is robust for different network operators and types.

C. *Robustness*

In this section, we evaluate the robustness of `PQSA-GR` under the following network environments.

(NC#1) Network Operators. We observed from Figure 7 and 8 `PQSA-GR` performs superiorly and consistently across different operators and types, despite being trained using the same set of datasets (dataset #1 to #4). This strongly suggests that `PQSA-GR` is spatially robust in the sense that algorithms trained using a dataset with a sufficiently wide spectrum of network bandwidth, can perform consistently over wide range of network environments regardless of operator, type, *etc.*.

(NC#2) Locations. To complement the result in Figure 1 we evaluated the QoS achieved by `PQSA-GR` in *three* locations

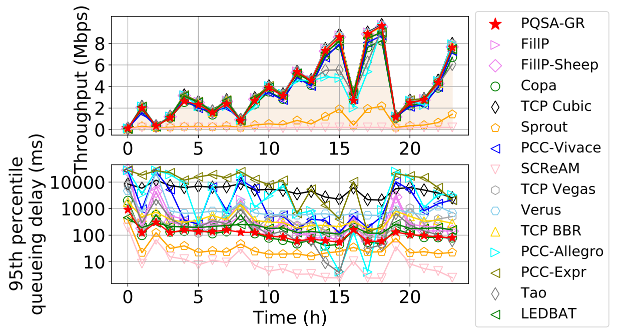

Fig. 11. **The hourly mean QoS of CCAs.** *We showed that* PQSA-GR *is temporal robust, as it tracks the target delay and bandwidth variations closely.*

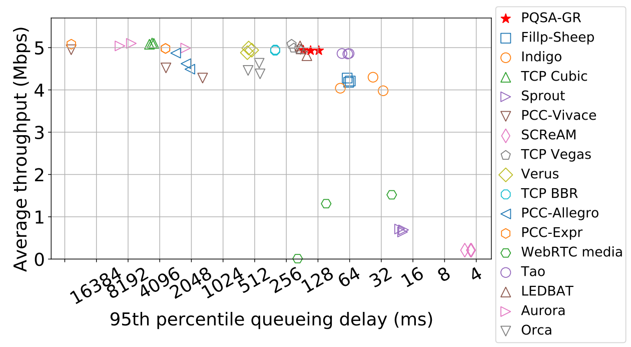

Fig. 12. **QoS of CCAs for three session durations.** *We showed that* PQSA-GR *achieves the consistent QoS for session duration* 30*s*, 60*s, and* 120*s.*

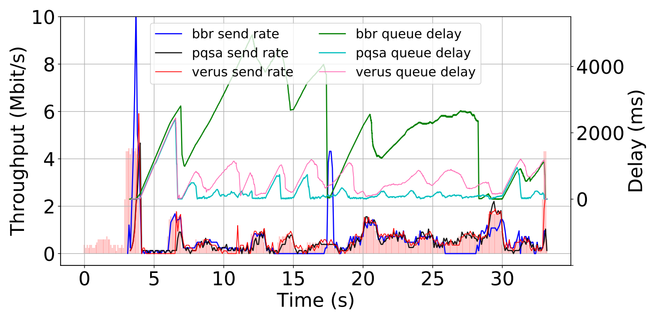

Fig. 13. **Comparison of rate adaptation behaviors in low throughput level 0**

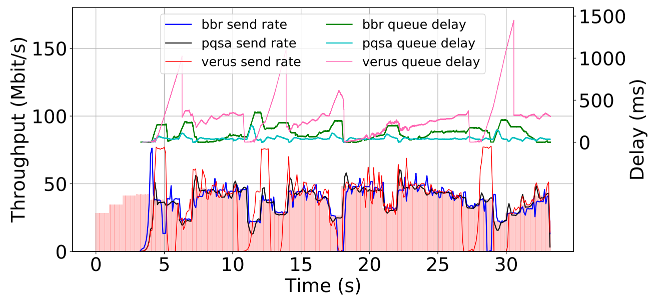

Fig. 14. **Comparison of rate adaptation behaviors in high throughput level 35**

using datasets from #2 to #4, respectively. Figure 10 shows that PQSA-GR performs consistently in three locations compared to the others, *i.e.*, PQSA-GR tracks the target delay across locations while maintaining comparable throughput, which further validates its spatial robustness.

(NC#3) Propagation delay and (NF#1) Buffer size. As shown in Figure 2 and 3, PQSA-GR is insensitive to propagation delay and link buffer size changes because PQSA-GR adopts a delay-based CCA approach. Parameter T (*i.e.*, queueing delay threshold) controls the length of the queue, which has no connection with buffer size and propagation delay.

(NC#4) Time of Day. We consider temporal robustness – QoS variations over time (*i.e.*, hours). We classified datasets of #2 to #4 by the hour of occurrence and ran emulated experiments in the online adaptation phase with trace data from each hour respectively to obtain the hourly QoS result. Figure 11 plots the hourly delay and throughput achieved by all CCAs. Again, we found that PQSA-GR exhibits consistent delay hourly, which is close to the target delay. In comparison, the delays achieved by the other CCAs vary far more substantially over time. To appreciate the variations in network conditions, we also plot the hourly mean bandwidth in Figure 11 (shadow color) along with the hourly mean throughput achieved. Despite the fact that the bandwidth trace data were captured in the same physical location by a stationary client using the same mobile network, the hourly bandwidth can still vary significantly from a low of $0.1Mbps$ to a high of over $9.1Mbps$. The throughput achieved by PQSA-GR also tracks the bandwidth variations closely. Overall, PQSA-GR's spatial and temporal robustness is significant because it substantially reduces the need to train algorithms for specific locations or to retrain algorithms periodically over time, as long as the dataset for offline training covers wide enough bandwidth variations.

(AB#1) Flow Duration. To complement the result in Figure 4 we show in Figure 12 that, for the same dataset, PQSA-GR achieves the best QoS with consistent delays for three online flow durations, *i.e.*, 30s, 60s and 120s. As the flow duration for the offline training phase is always 30s while the online adaptation phase can test different session durations, the result shows the benefit of parameter-set re-optimization in the online adaptation phase, while the others either cannot achieve consistent QoS, or perform worse than PQSA-GR.

D. Deep Dive

Microscopic view of PQSA-GR: To shed light on the superior results achieved by PQSA-GR, we compare its rate adaptation behaviors to BBR and Verus, which are a typical TCP-like CCA and learning-based CCA respectively. Figure 13 and 14 show the send rates and resultant packet queueing delays along with the bandwidth variations (shadow color), for both low (0) and high (35) throughput levels respectively. BBR has the same rate of adaptation aggressiveness for the two levels. Although this works well in high throughput levels with stable bandwidth, it causes disastrous queueing in low levels with extremely low bandwidth and substantial bandwidth variations. In contrast, PQSA-GR has different

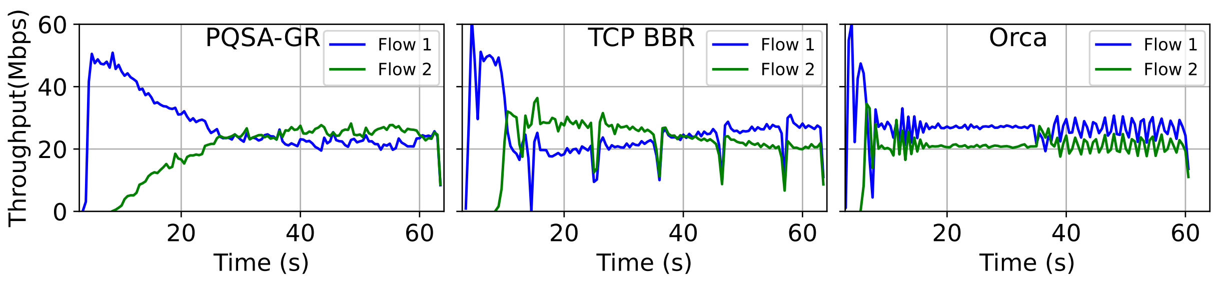

Fig. 15. **Convergence property of different CCAs**

TABLE IV
Flexible target delay control. *We show that PQSA-GR's actual delays do not deviate from the target ones.*

Target Delay (ms)	50	100	200
Actual delay (ms)	68.2	101.4	212.1
Throughput (Mbps)	12.1	14.52	15.0

TABLE V
Epoch and throughput quantization. *We show the performance impact of epoch and throughput quantization.*

Epoch (s)	30	60	120	240	30 (wo-$\mathtt{SD}_1$)
Actual Delay (ms)	100.1	71.9	52.5	57.5	45.6
Throughput (Mbps)	4.9	4.1	3.5	3.6	3.8

aggressiveness for different levels: for low levels, it adapts to the bandwidth by always having a lower-than-bandwidth send rate, increasing it more gently and decreasing it rapidly, thus it has a smaller send rate than the other two and results in a lower delay most of time; for high levels, it changes to increase send rate more aggressively by learning that the bandwidth would remain stable for a longer time. Due to unpredictable network conditions in mobile networks, Verus cannot learn a stable RTT-to-send-rate profile, thus resulting in unexpected rate adaption behaviors.

Flexible QoS objective control: PQSA also offers, for the first time, a tool for content/service providers to explicitly and flexibly control QoS objective via target delay. To evaluate this capability we show in Table IV that, for the three target delays (*i.e.*, $\rho = 95$, $Q = 50, 100, 200ms$), PQSA-GR achieves the actual delays reasonably close to the target ones. By having a stringent target delay, *e.g.*, $50ms$, PQSA-GR tradeoffs a lower bandwidth efficiency.

Epochs: Another interesting direction is to investigate the relative performance contributions by key components or techniques of PQSA. We evaluate the performance impact of parameter-set re-optimization with different epochs. Specifically, we run a 240s PQSA-GR flow over dataset #2 to #4 and apply parameter-set re-optimization every 30s (default), 60s, 120s, and 240s (which represent no parameter-set re-optimization). Table V shows that, with longer epoch, queueing delay is reduced at the sacrifice of bandwidth efficiency, and PQSA-GR with default epoch can achieve the best QoS. Due to long timescale bandwidth variations, the parameter-set trained from PQSA cannot be effective for longer session duration, which tends to result in bandwidth underutilization.

Throughput quantization: We study the impact of removing throughput quantization in offline training phase, indicated by wo-$\mathtt{SD}_1$ in Table V. *i.e.*, PQSA takes all traces as a single throughput level. It is clear that the throughput quantization is essential to PQSA: PQSA-GR without throughput quantization (30s wo-$\mathtt{SD}_1$ in Table V) has the worse QoS than the others with it, as a single parameter-set cannot capture the correlation between bandwidth fluctuations and QoS.

Fairness: PQSA-GR intrinsically is one of the delay-based TCP, thus it inherits similar fairness as the others like BBR. We validate it by running two flows together using PQSA-GR, BBR and Orca at a constant bandwidth of 48Mbps, where flow 1 starts first and lasts for 60s, and flow 2 starts after 5s and lasts for 55s. As shown in Figure 15, it takes a shorter convergence time for PQSA-GR and BBR to reach the fair share, compare to Orca, when the second flow enters the network.

CPU Overhead: Since PQSA-GR is a TCP-like CCA, it results in little CPU overhead as the other TCP-like CCAs [14], [32]. In contrast, the CPU overhead of learning-based CCAs is significantly greater than that of TCP-like, as we measured, the CPU overhead of Indigo is 11.76%, Orca is 3.37%, and PCC-Expr is 97.78%.

VI. Conclusion

In this work, we argue that current approaches to redesigning the transport layer to include learning-based CCAs are significantly disruptive and result in suboptimal transport protocols. Instead of redesigning the transport layer, we argue for (1) modifying the transport layer to externalize key parameters and (2) then determining optimal parameters for network states with a data-driven framework PQSA. Our design choices ensure that PQSA-optimized GR, is immediately deployable, and provides optimal and flexible QoS performance to users like content/service providers.

VII. Acknowledgment

This work was supported by Beijing Municipal Natural Science Foundation (No. 4212028), National Natural Science Foundation of China (No. 62072439 and 62090020), National Key Research and Development Plan of China (No. 2022YFB4500400), Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA0320300), Shandong Excellent Young Scientists Fund Program (Overseas)(No. 2023HWYQ-045), Shandong Provincial Natural Science Foundation (No. ZR2019LZH004), and HKSAR RGC General Research Fund (GRF/14206521). Corresponding author: Ke Liu (liuke@ict.ac.cn).

REFERENCES

[1] Fillip. https://github.com/fillthepipe/fill-the-pipe.

[2] Fillip-sheep. https://github.com/fillthepipe/fillp-sheep.

[3] PCC-Expr. https://github.com/PCCproject/PCC-Uspace.git.

[4] SMT Bandwidth Trace. https://github.com/ACS-DNET/PQSA-GR-Traces.git.

[5] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. SIGCOMM 20, page 632–647, 2020.

[6] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *SIGCOMM 18*, 2018.

[7] H. T. Alvestrand. Overview: Real time protocols for browser- based applications, November 2016.

[8] Jeffrey G. Andrews, Stefano Buzzi, Wan Choi, Stephen V. Hanly, Angel Lozano, Anthony C. K. Soong, and Jianzhong Charlie Zhang. What will 5g be? In *JSAC 14*, volume 32, pages 1065–1082, 2014.

[9] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *NSDI 18*, 2018.

[10] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *SIGCOMM 94*, page 24–35, 1994.

[11] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):20–53, October 2016.

[12] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. Pcc: Re-architecting congestion control for consistent high performance. In *NSDI 15*, 2015.

[13] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-learning congestion control. In *NSDI 18*, Renton, WA, 2018.

[14] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. A unified congestion control framework for diverse application preferences and network conditions. In *CoNEXT 21*, 2021.

[15] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing cdn-isp collaboration to the limit. *SIGCOMM Comput. Commun. Rev.*, 43(3):34–44, July 2013.

[16] Sangtae Ha and Injong Rhee. Taming the elephants: New tcp slow start. *Comput. Netw.*, page 2092–2110, June 2011.

[17] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[18] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *ICML 19*.

[19] Ingemar Johansson. Self-clocked rate adaptation for conversational video in lte. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, page 51–56, 2014.

[20] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The quic transport protocol: Design and internet-scale deployment. In *SIGCOMM 17*, 2017.

[21] W. K. Leong, Y. Xu, B. Leong, and Zixiao Wang. Mitigating egregious ack delays in cellular data networks by eliminating tcp ack clocking. In *ICNP 13*, 2013.

[22] Wai Kay Leong, Zixiao Wang, and Ben Leong. Tcp congestion control beyond bandwidth-delay product for mobile cellular networks. In *CoNEXT 17*, 2017.

[23] Xu Li, Feilong Tang, Jiacheng Liu, Laurence T. Yang, Luoyi Fu, and Long Chen. Auto: Adaptive congestion control based on multi-objective reinforcement learning for the Satellite-Ground integrated network. In *ATC 21*, pages 611–624, 2021.

[24] K. Liu and J. Y. B. Lee. On improving tcp performance over mobile data networks. *IEEE Transactions on Mobile Computing*, 2016.

[25] Y. Liu and J. Y. B. Lee. An empirical study of throughput prediction in mobile data networks. In *GLOBECOM 15*, 2015.

[26] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. Multi-objective congestion control. In *EuroSys 22*, 2022.

[27] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom 01*, 2001.

[28] Tong Meng, Neta Rozen Schiff, P. Brighten Godfrey, and Michael Schapira. Pcc proteus: Scavenger transport and beyond. In *SIGCOMM 20*, 2020.

[29] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. Ledbat: The new bittorrent congestion control protocol. In *ICCN 10*, 2010.

[30] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In *SIGCOMM 14*, 2014.

[31] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *SIGCOMM 16*, 2016.

[32] Han Tian, Xudong Liao, Chaoliang Zeng, Junxue Zhang, and Kai Chen. Spine: An efficient drl-based congestion control with ultra-low overhead. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 261–275, 2022.

[33] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI 13*, pages 459–471, Lombard, IL, April 2013.

[34] 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 *ATC*, 2018.

[35] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *SIGCOMM 15*, 2015.

[36] Guanghui Zhang and Jack Y. B. Lee. Ensemble adaptive streaming – a new paradigm to generate streaming algorithms via specializations. *IEEE Transactions on Mobile Computing*, 19(6):1346–1358, 2020.

[37] Guanghui Zhang, Ke Liu, Haibo Hu, and Jing Guo. Short video streaming with data wastage awareness. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2021.