Learning Production-Optimized Congestion Control Selection for Alibaba Cloud CDN

Xuan Zeng[1*], Haoran Xu[2*], Chen Chen[1*], Xumiao Zhang[1], Xiaoxi Zhang[2], Xu Chen[2], Guihai Chen[3]
Yubing Qiu[1], Yiping Zhang[1], Chong Hao[1], Ennan Zhai[1]
[1]*Alibaba Cloud* [2]*Sun Yat-sen University* [3]*Nanjing University*

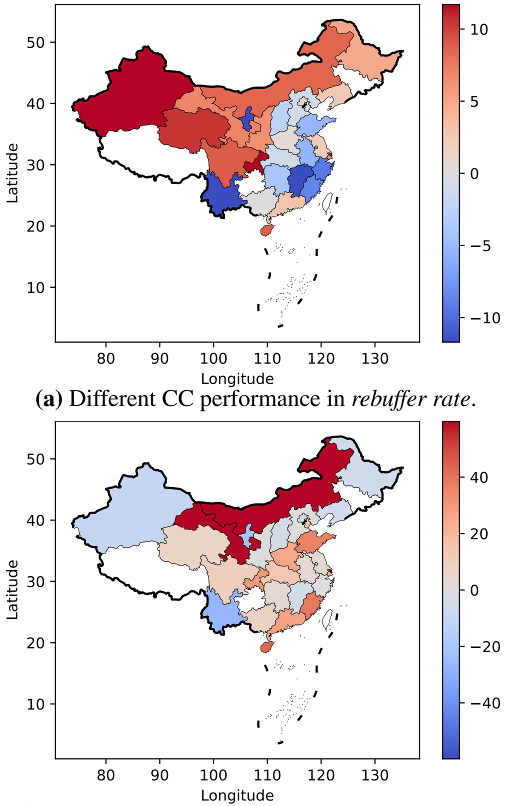
Abstract

Today's content delivery networks (CDNs) typically use static congestion control (CC) configurations, yet the diverse network environments preclude a universally optimal CC for all geographical regions, as evidenced by our extensive measurements. Current CC algorithms, limited by narrow applicability or high maintenance costs, struggle in large-scale CDNs. This work introduces AliCCS, the first CC Selection (CCS) approach tailored for production CDN, integrating fine-grained domain knowledge for learning to choose the best CC from existing, well-established ones. Through an over-one-year real-world deployment in Alibaba Cloud CDN, AliCCS has enhanced the Quality-of-Experience (QoE) by up to 9.31%, surpassing the competitive margin in the CDN market, and significantly reduced the retransmission rate by 25.51% to 174.36% across all provinces of China, leading to cost savings over 10 million US dollars. We also share key insights and experiences from deploying AliCCS at scale, highlighting traffic patterns in Alibaba Cloud CDN.

1 Introduction

Content Delivery Networks (CDNs) serve as the invisible backbone of the Internet, responsible for carrying over 70% of global traffic [4]. Enhancing Congestion Control (CC) in CDNs can significantly improve user experience by directly tackling network efficiency and reliability. It also achieves considerable cost savings for CDN operators, by offering a simpler, more resource-efficient alternative, compared to network or application layer modifications. Therefore, as a CDN provider, who has the greatest control and clearest visibility at the transport layer where CC is central, optimizing CC enables us to make more impactful changes that ISPs or application providers might not readily achieve (§2).

Currently, most CDNs still adopt the one-size-fits-all CC configuration [16,32] across their geographically distributed regions, yielding a suboptimal overall QoE due to the vast diversity of CDN network environments. Conventional approaches either *fine-tune the parameters* of one type of CC or *invent a new CC*, targeting only specific network scenarios. Our experience shows that leading CC solutions exhibit significant variations in user QoE for content delivery services. No single CC scheme can always be the optimal choice across all regions of our CDN network, *e.g.,* ranging from rebuffer rate (Figure 1a) to bandwidth cost (Figure 1b)[1]. Given this complexity and variability, we raise a question: *How can we accurately select the best CC among well-established ones for distinct CDN nodes under dynamic network conditions?*



(a) Different CC performance in *rebuffer rate*.

(b) Different CC performance in *bandwidth cost*.

Figure 1: Performance comparison across 29 provinces in China. Red (Blue) means CUBIC (BBR) achieves better performance. A darker color indicates a bigger performance gap.

Machine learning (ML), with its widely proven proficiency in learning from diverse data, is particularly suited to address the dynamic and heterogeneous nature of CC Selection (CCS). It can adapt to varying CC performance across different factors. Despite this, extensive research on modifying CC, including learning-based approaches [14,26,45,47,49,52], have not fully met the unique and stringent needs of production CDNs. From our experience, applying ML to CCS for large-scale production CDN presents the following key challenges, directly stemming from critical CDN requirements:

[*] These authors contributed equally to this work.

[1] Details on the fair A/B test of CUBIC and BBR are in Appendix §A.

Table 1: Comparison of AliCCS with state-of-the-art ML-based Congestion Control Selection methods.

	Optimization Goal	**Granularity**	**Satisfy CDN req.**[I]	**Scales of CDN nodes, users/requests**
Sage [49]	Throughput/latency related	N/A[II]	No	N/A
Disco [47]	Throughput/latency related	Per-connection	No	N/A (<10 servers and clients emulated)
Configanator [32]	Web performance (PLT)	Per-Network-Class	Yes	3 Global CDN PoPs, 8.2M requests
AliCCS	Rebuffer rate, latency	Per-connection	Yes	>1000 CDN nodes, 500M requests

[I] CDN requirements refer to those mentioned in §1: 1) scalability & generalizability, 2) interpretability, 3) managing inference overhead.
[II] N/A stands for Not Available.

- **Scalability and generalizability challenges.** Our CDN spans various geographical locations and network environments, necessitating distinct models for different nodes. However, managing individual models for the thousands of nodes in our global CDN is impractical due to extensive maintenance requirements. Moreover, most ML algorithms assume independent and identically distributed (i.i.d.) data between training and test domains, which may not hold true in practice. Developing a unified ML model that can efficiently scale and generalize across diverse data environments remains a crucial yet challenging goal.
- **Enhancing interpretability and accountability for deploying ML-based CCS.** Standard statistical tools alone do not suffice to evaluate ML models accurately due to potential sample selection bias and confounding factors [35]. It is therefore essential to integrate deep domain knowledge from our CDN with these tools to thoroughly assess the model's effectiveness and increase its interpretability before a full-scale CCS deployment in our production CDN.
- **Managing ML inference overhead without compromising CCS performance.** High performance of ML models often requires complex inference logic, leading to significant processing delays. Our real-world measurements reveal that non-optimized per-connection ML inference, *executed serially*, incurs significant overhead and undermines the queries per second (QPS) performance (refer to Table 3). Enhancing run-time efficiency with reduced ML inference delays is therefore vital to cost-effective CCS.

These challenges are not fully addressed by existing learning-based CCS efforts, summarized in Table 1. They do not incorporate the requirements and domain expertise of realistic large-scale CDN networks. Moreover, their reliance on methods like reinforcement learning [47,49] or online learning [32] necessitates handcrafted reward functions and immediate QoE feedback, both of which are impractical for the short video services supported by our CDN based on our experience.

To address the above challenges simultaneously, we introduce AliCCS, the first domain-specific, ML-based CCS scheme, tailored to *short video delivery*, a dominant workload in Alibaba Cloud CDN. AliCCS dynamically selects the most effective CC for each connection[2], aiming to (1) boost short video QoE, by reducing rebuffer rates and start-up delays, and (2) lower bandwidth cost, *e.g.,* retransmission ratio.

In developing AliCCS, we make the following contributions:

Firstly, it is crucial to define the right features and labels from usable data for ML models that predict key factors to optimize CCS, scale to numerous CDN nodes, and generalize to new environments. We reveal an observation (detailed in Figures 3 and 4) that network types decisively influence the best CCS. Therefore, our goal is to effectively learn a mapping from end-to-end TCP statistics to network types. However, this relation is *unstable* due to hidden states [35] like underlying network conditions. We then propose a decomposed Causal Graphical Model (CGM) using Generative Adversarial Networks (GAN) and domain knowledge, extracting features *that are invariant to the hidden states*. Additionally, we develop a hierarchical clustering algorithm to group a vast number of IP prefixes for model scalability improvement.

Secondly, to enhance the model's transparency and interpretability, we employ knowledge distillation to transfer insights from our complex, well-trained deep neural network (DNN) into a compact decision tree. This tree, developed using training data enhanced by the predictions of our DNN model, effectively replicates the DNN's decisions in a format that is easier to interpret. When we encounter performance degradation during online CCS inference, we substitute our large DNN model with this decision tree. This allows for precise tracking of any suboptimal predictions by identifying the specific features contributing to the inefficiency, providing better interpretability of our ML-produced CCS solutions.

Thirdly, we significantly reduce model inference overhead by constructing an IP-prefix based cache that leverages temporal and IP-prefix-specific similarities in the domain of the best CCS and network types. AliCCS has achieved 64.30× improvement in latency and 2.42× improvement in QPS for online CCS inferences. Moreover, we realize fine-grained CCS by implementing it in a per-connection manner. This also bypasses the complexity and QoE sensitivity issues in selecting the grouping granularity, which is inherent in existing CC control methods that rely on request grouping [32].

Deployment and experience. AliCCS has served Alibaba Cloud CDN's production for over one year. We train our model on labeled data from App #1[3], one of the leading short video companies in the world, obtained from about 30% of the CDN nodes supporting its services. The well-trained model serves online CCS inference in an over-one-year in-production deployment for both App #1 and App #2,

[2] Multiple requests can multiplex on each CDN connection.

[3] Partner apps are anonymized due to NDA of our company.

two major providers in the global short video market. AliCCS achieves 95.8%–99.0% Wi-Fi/4G prediction accuracy in about 400 nodes of three main ISPs in China and Southeast Asia, covering almost all provinces in China, demonstrating our ML model's generalizability. Combining our model's predicted network access type and the mapping to CC choices, we reduce the rebuffer rate by 4.76% and retransmission bandwidth by 25.51%–174.36%. In terms of run-time efficiency, AliCCS achieves CPU usage below 0.79% (at most two cores per CDN node) and maintains the minimum memory consumption below 2.90 GB (see Figure 11). In addition, it achieves 64.32× processing delay reduction and 2.42× maximum QPS improvement (see Table 3).

2 Background and Motivation

In this section, we detail our motivation for employing CCS in Alibaba Cloud CDN, focusing on short video services. We then share observations that drive AliCCS's design choices.

2.1 Overview of CDN at Alibaba Cloud

Alibaba's CDN spans over 70 countries with a total egress bandwidth exceeding 180 Tbps that serves over 120 EB (*i.e.,* 120×2^{60} Bytes) of data every year. In particular, it has 2800+ nodes in China and South Asia, covering a wide range of geographical regions and ISPs, where AliCCS is deployed. By caching content at edge servers, the CDN significantly reduces back-to-origin bandwidth and enhances user proximity. It offers diverse services, ranging from large file transfers to media streaming services, among which short video services take the most significant share of traffic demands.

CDN providers aim to deliver high-quality services cost-effectively, focusing on improving *user QoE* and reliability to stand out from competitors. *Bandwidth costs* constitute a major portion of operational expenses, which means even slight and relative reductions in data transfer can lead to significant absolute savings at scale.

2.2 Motivation: CCS for Short Video Services

Short videos, *e.g.,* supported by App #1 and App #2, have become a major workload of Alibaba Cloud CDN. The CDN should then be optimized to deliver short videos with high user QoEs while incurring minimum bandwidth costs.

QoE of short video delivery. Efficient content delivery for short videos in CDNs necessitates quick video start times and smooth playback. Traditional video streaming focuses on metrics like bitrate, resolution, and loading appropriate initial buffering video segments (typically 2-10 seconds) to ensure high-quality, continuous playback over longer durations. In contrast, short video services emphasize minimizing the rebuffer rate to reduce stalls and the initial buffering delay, typically less than 1 second for the first frame to load.

Why adjust CC? Extensive efforts have been made to improve video delivery QoE by refining application layer strategies like Adaptive Bitrate (ABR) algorithms and content pre-fetching. However, these are beyond the control of CDN providers and thus not considered in this work. In large-scale production CDNs, optimizing CC directly targets network bottlenecks, leading to immediate improvements in content delivery, especially for short video services. Besides, adjusting CC offers a straightforward and resource-efficient alternative to overhauling application layer configurations, including fine-tuning ABR [7,30,43] and content pre-fetching [17,29] settings, or transitioning from HTTP/2 to HTTP/3 [5,42].

Table 2: Performance comparison of different CCs for 4G networks (*CUBIC Agg.* means CUBIC tuned to aggressively trigger retransmissions to lower rebuffer rates).

CC Algorithm	CUBIC	CUBIC Agg.	BBR
Video Rebuffer Rate	1.13	1.05	1.0
Retransmission Rate	1.41	1.53	1.0

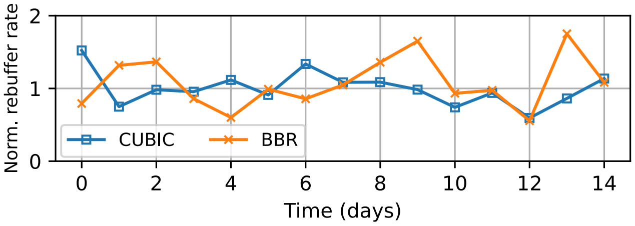


Figure 2: A two-week A/B test comparing BBR and CUBIC, serving short videos for App #1 in one CDN node.

Why CCS? While conventional approaches generally focus on parameter fine-tuning (*e.g.,* Bayesian Optimization) for a given CC [8], our experience has shown that the benefits of such tuning can be marginal in certain scenarios. This is because each CC algorithm has inherent limitations in how it identifies and reacts to congestion, which may fall short in situations violating their assumptions. For instance, in tuning CUBIC for 4G networks, a substantial increase in sensitivity to trigger retransmissions is necessary to match BBR's rebuffer rate in 4G. This adjustment, however, leads to an increase in the retransmission rate from 1.41 to 1.53 (see Table 2). Compared to the alternative of *inventing new CC algorithms*, our approach of selecting the best among long-living CC schemes, known for their effectiveness in variable scenarios, *provides robust worst-case performance*. It reduces potential revenue loss associated with inadequately tested CCs.

Why ML for CCS? Due to dynamic network conditions and user traffic in CDNs, *rule-based* CCS approaches are generally impractical. This is supported by our measurements showing volatile patterns of CC performances across both regions (Figure 1)[4] and time (Figure 2). Moreover, our choice of using ML for CCS is driven by our observations and domain knowledge, based on which CCS can be reduced to a classification problem, elaborated in §2.3. This problem reduction directly informs our model design, where we integrate supervised

[4] We find regional proximity does not yield network metrics' similarity, likely due to diverse traffic policies (*e.g.,* rate-limiting) in nearby regions.

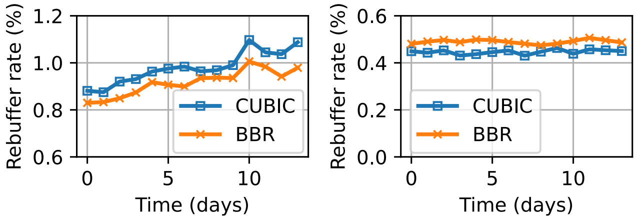


Figure 3: Contrasting advantages of BBR and CUBIC varying network types, *i.e.,* 4G (left) and Wi-Fi (right).

learning with fine-grained domain knowledge. Consequently, other ML choices, such as reinforcement learning [47, 49, 52] and no-regret (or online) learning [14, 32, 50] do not align well with our needs. They are in general tailored to sequential processes where data are gradually available, adding unnecessary complexity and inefficiency to CCS.

2.3 Key Observations

Our goal is to develop an ML system that selects the best CC algorithm from a set of candidates to enhance user QoE and reduce bandwidth costs for short video services across extensive CDN nodes.

Observation I: CCs' advantages vary regionally and temporally. Despite extensive efforts to optimize the performance of individual CC algorithms, we find that no single algorithm consistently outperforms others in all scenarios. Our Figures 1a and 2, illustrate that the superiority of BBR and CUBIC in rebuffer rate varies across regions and fluctuates over time, demonstrating that using a fixed CCS rule is generally impractical. This observation aligns with recent findings [49], which report similar fluctuations in the CC superiority.

Observation II: Network types decisively influence CC advantages. We compared short video rebuffer rates between BBR and CUBIC over 14 days, presented in Figure 3. Both groups, represented by the blue and orange curves respectively, use the same network type, *i.e.,* 4G for the left figure and Wi-Fi for the right. This reveals a strong correlation between network access and the best CC choices: BBR outperforms in 4G connections (left) while CUBIC wins under Wi-Fi (right). Other CCs like Vegas [11], Reno [23], and Westwood [40], exhibited significantly inferior performances (*e.g.,* 30%–40% higher rebuffer rate than CUBIC and BBR) for short video delivery in our production CDN, based on comprehensive pre-deployment measurements. To ensure performance reliability and stability, we choose CUBIC and BBR as the CC candidates. Moreover, we examine the impact of different features on CC selection. Figure 4 shows that network type has the highest information gain [3], vastly outweighing other features like retransmission rate and hour-of-the-day with information gain of 0.001 and 0.005 respectively[5]. This observation leads us to reduce the CCS problem by focusing on predicting the network type for each TCP connection, as it provides the most substantial predictive power of the best CC and keeps system complexity manageable. We then deterministically choose CUBIC for Wi-Fi and BBR for 4G to optimize short video QoE.

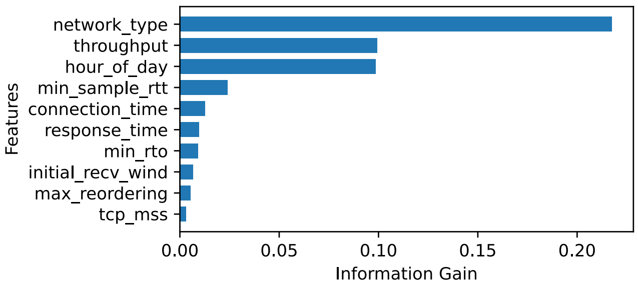


Figure 4: Top 10 features on our CDN with the highest information gain for predicting the best CC candidate. These features span four categories: transport layer (TCP) statistics (*e.g.,* retransmission rate and throughput), application layer statistics (*e.g.,* response time), link layer statistics (*i.e.,* network type) and timing (*i.e.,* hour of the day).

Observation III: ISP rate-limiting impacts CCs differently. Based on Observation II, we hypothesize that the contrasting performance of CUBIC and BBR varying network types is attributed to the ISP's traffic policies, particularly token bucket rate-limiting that may cause premature packet losses[6], which CUBIC misinterprets as congestion signals, significantly reducing throughput. To confirm this, we first utilize BBR's bandwidth probing mechanism to assess rate-limiting strength over months and then examine how CUBIC and BBR respond. We observe that 4G's rate-limiting generally increases from the start towards the month's end. During this, CUBIC's average throughput drops to less than 10% of that without rate-limiting (1472 Kbps vs. 8728 Kbps), and its average retransmission rate surges to 73× higher than its non-rate-limited performance. It highlights that CUBIC struggles in 4G rate-limiting conditions, while BBR's bandwidth probing technique can instead effectively identify the bandwidth constraints. Regarding BBR's inferior performance on Wi-Fi, it is primarily due to Wi-Fi's frame aggregation mechanism, which undermines BBR's bandwidth probing capabilities [18].

Observation IV: Network types need to be predicted dynamically. Obtaining network type via client reporting is impeded by software upgrade overhead and unreliable client data. Notably, labeled data from client accounts for only 5%–10% of total requests (see labeled data acquisition in §3). Additionally, private IP database solutions like MaxMind [2] offer limited coverage and accuracy of network type labels for our CDN network. Moreover, we observe that with 90% probability, a /24 IP prefix will significantly shift its proportion of dominant type (Wi-Fi or 4G) in 15 hours, necessitating dynamic network type prediction.

[5] Features like retransmission rate are analyzed but not shown in the figure, as they ranked outside the top 10.

[6] Packet losses can be categorized as transmission, congestion (from link contention), and rate-limiting loss. While the first two vary over time within the same network, the rate-limiting loss is consistently bound to 4G networks.

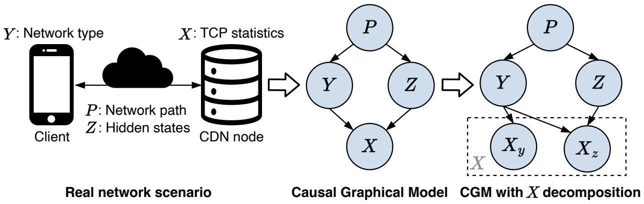


Figure 5: Illustration of decomposing features via CGM.

3 Design and Implementation

In this section, we outline the design of a unified ML model for all CDN nodes, enabling stable mapping from TCP statistics (additional features are listed in Appendix §B) to network types. One naïve solution would be to collect training data from all regions and train a comprehensive model. However, this method raises two significant issues: (1) The QoE performance in worst-case scenarios is not guaranteed, which is critical for production CDNs. (2) Regions that are underrepresented in the training data can impede model inference performance due to limited generalization. Essentially, weak network conditions and regions with fewer requests are often insufficiently represented in the training set. Training the model naïvely without eliminating the effects of these diverse network conditions, which are represented as underlying network characteristics, may lead to overfitting on good conditions or regions with ample requests. This, in turn, degrades model accuracy in poor network conditions or small regions, as evidenced in §5.4, undermining its capability to provide worst-case guarantees. Another straightforward CCS approach is to directly adopt the network type labels from datasets and then choose CUBIC for Wi-Fi and BBR for 4G. However, network types in the datasets are often unreliable and thus need to be predicted dynamically (refer to Observation IV).

3.1 Model Design and Training Method

Prior research on applying ML to networked systems often overlooks model generalizability under volatile network environments. In contrast, our model design aims to learn causal relationships between TCP data and network types[7], guided by our key observations (§2.3). This approach enhances prediction accuracy in new environments, crucial for generalizability. However, our measurements reveal that the relationship between TCP statistics and network type is generally *unstable*. This instability likely stems from end-to-end TCP statistics being influenced by network types as well as varying path-specific characteristics, such as buffer sizes and the number of competing flows across CDN nodes and IP prefixes.

Design overview. To tackle the instability in learning, our idea is to decompose the raw data features, which are TCP statistics ($\triangleq X$), into two parts, denoted by X_y and X_z respectively in Figure 5: one influenced solely by network types ($\triangleq Y$) and the other influenced by both Y and hidden state ($\triangleq Z$). Our goal is to create a model based only on features X_y for model stability. To do so, we construct a decomposed Causal Graphical Model (CGM) and tease apart the impact of network types Y and hidden states Z on TCP statistics X, by leveraging the one-to-one mapping between network paths ($\triangleq P$) and IP address prefixes observable on CDN servers. This ensures a stable model that maps X to Y while remaining *causally invariant to* Z. Finally, we incorporate domain knowledge as constraints and regularization terms, to ensure model scalability and reduce the noise.

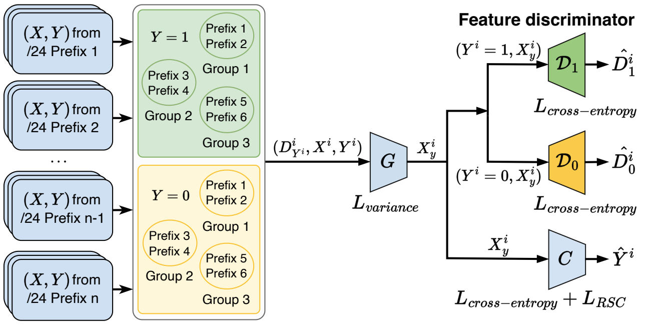


Figure 6: Overview of our ML model design.

Decomposing features via CGM. Ideally, to develop a generalized model for the diverse CDN nodes, we need to identify features for any TCP connection that have a stable relationship with our target Y. However, directly using X, the measured TCP statistics (*e.g.,* maximum congestion window size), is insufficient due to their dependence on both the network type Y and varying hidden factors Z (*e.g.,* path capacity), as depicted in the middle CGM in Figure 5. Different CDN nodes involve various paths P (interchangeable with connections) and thus different hidden factors Z, causing instability in the X and Y relationship. To address this, we leverage our decomposed CGM with X_y and X_z, the decomposed features of X (refer to Figure 5), and a condition satisfied by X_y: $Pr(X_y|Y,P) = Pr(X_y|Y)$. Therefore, our goal is *to identify features X_y for each connection that maintain a constant conditional probability distribution $Pr(X_y|Y)$ across all connections*. By training a classifier with these feature-target pairs $\{X^i_y, Y^i\}^m_{i=0}$, with a total of m samples, we can create a model that performs reliably on every CDN node. This is realized by integrating this decomposed CGM into a GAN, where a generator is trained to obscure the discriminator's ability of predicting the path from which the sample originates.

Learning path-invariant features via GAN. Through the above mathematical insights rooted in our decomposed CGM, learning a stable mapping from X to Y is converted to learning a path-invariant feature representation X_y given Y. Furthermore, leveraging the one-to-one mapping relationship between IP prefix and Internet path[8], the goal becomes *learning*

[7] The minor case of non-1:1 mapping between connection and network type is not yet addressed. Potential solutions are outlined in Appendix §C.

[8] The association between IP prefixes and Internet paths stays stable for several hours, despite occasional changes due to load balancers.

an IP-prefix-invariant feature representation for any specific network type. To achieve this, we adopt the GAN framework and construct two discriminators, marked by $\mathcal{D}_1$ and $\mathcal{D}_0$ in Figure 6, each outputting which /24-prefix group the sample comes from, for Wi-Fi ($Y = 1$) and 4G ($Y = 0$), respectively We then formulate two regularizers, R_{D_0} and R_{D_1}, based on the discriminators' predictions $\widehat{D}_0$ and $\widehat{D}_1$, corresponding to the ground-truth group labels D_0 and D_1, respectively[9]. These regularizers represent the classical cross-entropy-based classification loss of discriminators and are formulated as:

$$R_{D_0} = L_{cross-entropy}(D_0, \widehat{D}_0) \tag{1}$$

$$R_{D_1} = L_{cross-entropy}(D_1, \widehat{D}_1) \tag{2}$$

Scalability with domain knowledge-assisted clustering. Ideally, the outputs D_0 and D_1 correspond to the vectors indexing IP address prefixes. However, a scalability challenge arises: the discriminators need to distinguish which IP prefix each sample originates, among tens of thousands of candidates at each CDN node, causing intractable training complexity. To tackle this, we modify the discriminators' original outputs to be IP-prefix *groups* and introduce a domain knowledge-assisted clustering algorithm. As shown in Figure 6, we first separate the training data based on whether Y equals 1 or 0 and then adopt K-means clustering to group the IP /24 prefixes, with Euclidean distance adopted as the distance metric.

Labeled data acquisition. We collect labeled data through collaboration and coordination with one of the short video service's client end. The label of network type is piggybacked via http header extensions from client side. Note that due to the overhead associated with software development and client upgrades, the volume of labeled data obtainable from client side is limited between 5% and 10% of its total requests with limited region coverage in a limited period of time.

Feature selection. Our CDN spans nearly all provinces in China, handling numerous TCP connection features from user requests, requiring efficient feature selection due to significant data scale and diversity. We choose information gain as the selection criteria for its intuitiveness and broad applicability. However, a one-shot feature does not meet our production-level accuracy requirements. Therefore, we also periodically diagnose AliCCS's model performance by combining the knowledge-distilled decision tree and Shapley value analysis to tweak the feature sets, which will be elaborated in §3.2.

While the core of our ML model to predict network types, integrating a decomposed CGM with domain-specific GAN and clustering, it also demands meticulous handling of numerous corner cases. Below, we outline two additional techniques used in our production CDN for effective model training.

Diversified features for smaller ISPs. Through training with CDN data, we have learned that, while a universal feature representation generally works well, significant deviations in feature distribution exist, in particular for smaller ISPs compared to major ones. This discrepancy can cause a model to underperform with smaller operators. By identifying and transforming alternative features from smaller ISPs to align with the broader distribution, we can ensure a more robust and effective model. To enhance generalizability and avoid over-reliance on dominant features, our model employs the Representation Self-Challenging (RSC) method [22]. Unlike Dropout, which randomly deactivates features, RSC uses gradients to selectively mute predictive features, encouraging the model to utilize a broader range of features. Therefore, we integrate an RSC-based regularizer into our training loss, denoted as L_{RSC} (used later in Equation 5). This approach is particularly effective for data from smaller-scale ISPs, diversifying the model's focus and ensuring robustness across various scenarios that deviate from larger operators.

Noise reduction with domain knowledge. To reduce feature noises, we again utilize the domain knowledge that requests with the same IP /24 prefix usually follow identical IP paths. Thus, the extracted features from the same IP /24 prefix should align closely, sharing the same path-specific hidden states X_z and the desired X_y. We translate this insight into a regularizer $L_{variance}$. Let $\mathcal{S}_{prefix}$ denote the set of those X^j sharing identical IP /24 prefixes and $\overline{X}_{prefix}$ denote the average of $X^j \in \mathcal{S}_{prefix}$. We formulate $L_{variance}$, the sum of the variance of extracted features of each /24 prefix group, as follows:

$$L_{variance} = \sum_{prefix=i}^{n} (X^j - \overline{X}_{prefix})^2_{X^j \in \mathcal{S}_{prefix}}. \tag{3}$$

Training method. Utilizing Equations 1, 2, and 3, we construct the loss functions for training the discriminators, generator (G), and the classifier (C) according to the following:

$$L_D = R_{D_0} + R_{D_1}, \tag{4}$$

$$L_C = L_{cross-entropy}(Y, \widehat{Y}) + L_{RSC}, \tag{5}$$

$$L_G = L_{variance}, \tag{6}$$

$$L_{all} = L_C - \lambda_1 * L_D + \lambda_2 * L_G, \tag{7}$$

where L_D is the discriminator's classification loss, L_C is the classifier's classification loss, and L_{RSC} is a term inspired by RSC regularization. L_G represents the regularization term applied on the generator G. L_{all} is the overall training loss, with hyperparameters λ_1 and λ_2 balancing the trade-off between model fitting and generalization. Further details on the design trade-off can be found in Appendix §D. Adhering to the standard GAN training method, we alternate between two training phases. We initially train the discriminators using loss L_D, with fixed generator and classifier parameters. Subsequently, we train the generator and classifier using the combined loss L_{all}, while keeping the discriminators fixed.

[9] In practice, to eliminate CC algorithm bias and obtain IP-prefix-and-CC-invariant features, we can instead cluster data by the combination of CC and IP-prefix, and re-define the outputs of $\mathcal{D}_1$ and $\mathcal{D}_0$ to be the combinations. Or, we can add another discriminator for it.

3.2 Model Interpretability Enhancement

In addition to generalizability and scalability, it is essential to establish trust before deployment and understand the root cause of prediction errors. This is crucial for iterating and refining the model, especially during incremental deployment in a production CDN. Model interpretability entails two requirements: First, its behavior must be easily verifiable by operational engineers using their domain expertise. Second, it should accurately reflect the ML model's behavior. To this end, we have chosen a tree-based model, which satisfies these requirements and has proven to be particularly well-suited for networking problems that span local and global control [31].

Knowledge distillation with augmented features. We distill the knowledge of our GAN-based deep learning (DL) model into a decision tree-based model for performance diagnosis. We first input all features $\{X^i\}_{i=0}^m$ from the training set to the trained DL model to obtain the prediction probabilities $\{Y_0^i, Y_1^i\}_{i=0}^m$. Then, we combine them to create a distilled version of the training set, $\{X^i, Y_0^i, Y_1^i\}_{i=0}^m$. Next, we construct a multi-output regression decision tree and train it using the distilled training set. Each output in the tree corresponds to the predicted probability of one CC selection solution, *i.e.,* Wi-Fi and 4G. The utilization of feature-probability tuples (X^i, Y_0^i, Y_1^i) rather than the standard feature-label pairs (X^i, Y^i) reflects our domain-specific GAN-based model design, where we construct two discriminators for Wi-Fi and 4G network types respectively, leading to higher interpretability.

Using our decision tree in practice. The tree-based models enhance the robustness of ML models by enabling performance issue debugging and helping understand the model predictions to build trust. For instance, during the initial deployment phase, when encountering a low accuracy within a specific /16 IP prefix, we applied Shapley analysis [28] to the output of the decision tree for those examples. This combination allows us to quantify feature importance for predictions on subset of samples and understand the model predictions for certain IP prefixes. We then found that the model excessively relied on the TCP maximum segment size (MSS) feature from that specific /16 IP prefix. Therefore, we adjusted the model to reduce its reliance on MSS and broaden its feature usage, effectively boosting the accuracy. Beyond debugging, decision tree models also assist the production team in comprehending and gaining confidence in our ML model, which is essential for its deployment in production. It is crucial to note that the distilled decision tree is employed solely for performance diagnosis rather than online inference, for two primary reasons. First, our caching mechanism, discussed later in §3.3, effectively mitigates DL model's inference costs. Second, empirical observations reveal that the distilled model's accuracy is much lower than that of our DL model. Nevertheless, despite a 5%–7% accuracy drop, we find the distilled tree model still captures error-related features, by aligning its prediction with the DL model for most samples. For subsets with significant prediction discrepancies, we increase the tree depth and retrain the tree until the alignment passes a threshold.

3.3 Inference Pipeline at CDN Nodes

Integrating our ML model (§3.1) into the networking stack of CDN nodes poses two technical challenges: (1) The model inference time must be negligible compared to the normal request processing time to maintain high request processing rates (*e.g.,* approximately 10k QPS) for any CDN node. (2) Hardware costs need to be considered, as excessive use of additional CPU or GPU cores per CDN node can be impractical and costly, potentially outweighing the benefits. We develop an efficient inference pipeline to tackle these challenges.

Centralized vs. distributed inference deployment. To implement automatic and intelligent CCS across CDN nodes using a well-trained DL model, we evaluated two approaches: centralized model inference services and distributed model replication on each node. Empirically, the centralized approach offers more abundant hardware resources in a cluster to deploy the model, facilitating faster ML model inference. However, the network latency between the centralized cluster and geo-distributed CDN nodes can be prohibitively high, potentially causing millisecond-scale delays for serving each batch of requests. Therefore, we opted for the distributed approach, replicating and performing inference locally on each CDN node to achieve greater responsiveness and efficiency.

This decision to adopt a distributed approach leads us to an efficient inference pipeline, grounded by two key observations drawn from comprehensive empirical studies:

- **Temporal stability of the best CCS and IP-based caching.** Consistent with a previous study [47], we find that the best CC choices, influenced by network path characteristics, remain stable for several hours. Specifically, our measurement reveals that a /24 IP prefix stays dominated by either Wi-Fi or 4G type (purity over 90%) for more than 2 hours with 87% probability[10]. And it can remain dominated by one of these types for 15 hours or more with 10% probability. This stability allows us to cache inference results for requests from specific paths, enabling their reuse for subsequent requests from the same path and significantly reducing inference costs over time.
- **Consistent CC choices based on IP prefix similarity.** IP networks are typically organized in subnets, and IP routing is based on the longest prefix matching. As a result, requests from the same IP prefix often undergo similar path decisions at a given time. As shown in Figure 7, among the top 10K /24 prefixes with the most requests, 90% of them exhibit a *purity* (percentage of connections with the same network type) higher than 91%, and these prefixes account for 84% of the total requests seen by our CDN. This allows us to not only cache results for the last few hours but also aggregate the results of our predicted CC choices

[10] We infer these dynamics may stem from ISP's load balancing and management, but we lack visibility into ISP's network to validate such hypothesis.

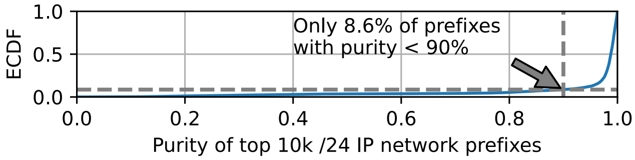


Figure 7: CDF of purity distribution of /24 prefixes.

under the same IP prefix into a single entry, leveraging their homogeneity in CC selection to save memory.

Inference pipeline. Figure 8 depicts the pipeline developed utilizing the above insights. We minimize the inference delay in the request processing pipeline by decoupling the process into online cache lookups (shown as the "Online processing pipeline" box) and periodic offline updates (shown as the "Offline processing pipeline" box). The latter records pairs of requests' IP addresses and the CC choices chosen given the predicted network types. The update is performed hourly to save computations, and also match the temporal stability of the best CCS. Every time a request server encounters a new client connection, it queries the mapping cache for the CC decision. This reduces the CCS latency from running a complex DL model inference to a lightweight cache lookup.

Efficient IP lookup cache. We implement an IP lookup cache using the Trie data structure [10,19,34,38,48], which provides O(1) lookup complexity with special hardware requirements. Instead of constructing a full Trie, which has high space complexity, we aggregate IP prefixes based on our previous observation on purity. We first construct a Trie at the /16 prefix level and examine the purity of each leaf node. Expansion stops if a node's purity surpasses a set threshold; otherwise, we expand until the node satisfies our purity standards or reaches the /32 prefix limit. A detailed algorithm is provided in Appendix §F, with further insights into threshold tuning available in Appendix §D. Even under /32 prefix granularity, requests of different network types are still observed. We speculate that the CDN only sees the public IP, which may be dynamically binded to various networks.

Empirical purity tuning. Tuning the aforementioned purity thresholds is crucial to model accuracy and space complexity. We adopt an empirical approach, starting with gathering online purity measurements, as shown in Figure 7. By analyzing the requests from the top 28 CDN nodes over one week, we find that only 8.6% of /24 prefixes have purity lower than 90%. By fine-tuning the threshold to aggregate IP addresses based on this observation and the available resources in each CDN node, the number of entries in the mapping cache is brought down from hundreds of thousands (*i.e.,* 10^5, the number of IP addresses seen by a CDN node in the last hour) to hundreds (*i.e.,* the number of /16 or /24 IP prefixes), resulting in a 614× memory saving without sacrificing prediction accuracy.

4 Experiences and Discussion

In this section, we discuss our deployment experiences and lessons learned in our real-world production CDN.

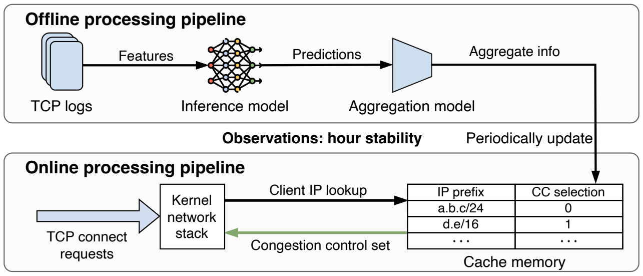


Figure 8: Inference pipeline at CDN nodes.

4.1 Deployment Experiences

ALICCS has served Alibaba Cloud CDN's production for over one year, supporting short video services for two major providers, spanning nearly all provinces in China. It reaches high network type prediction accuracy in around four hundred nodes of three main ISPs in China and Southeast Asia. We share some experiences in deploying ALICCS at scale.

Service-aware CCS. CDN requires tailored CCS strategies for different customers and services. This means that CCS can not be operated only in the kernel space where we are not aware of the service types and can not make customized decisions for different services. To better integrate with information from services and maintain flexibility, we decide to move AI processing mostly to user space.

TCPe (TCP extension) is our modular and customized TCP protocol stack implemented in the kernel space. Upon receiving each connection request from a client, it sends a query for the right CC decision to the mapping cache lookup using the client's IP address and then applies the query result of the CC algorithm accordingly in the kernel space. It also acquires TCP statistics from kernel space at a fine granularity, provided to the Log Server, which is responsible for collecting TCP statistics of CDN requests and send them to AI model for inference. We use the standard TCP socket's programming interface (*e.g.,* setsockopt) to set the CC algorithm.

AI Server, where the ML model resides, is not required to provide low latency, as it is decoupled from the online CCS service due to our designed Mapping Cache. We export the model trained in Pytorch in PMML format and load it into the AI server using the high-performance CPMML library. It continuously outputs predictions of network types using TCP logs from the Log Server, converts the predictions into CCs, and forwards them to the Aggregation Module.

4.2 Lessons Learned

We now share our lessons learned in deploying ALICCS.

What influences the QoE disparity of short video services under various CCs? Traditional CC design typically emphasizes maximizing throughput; however, our studies indicate that preventing frequent speed fluctuations is more critical, especially for short video services. Thus, while most CC mechanisms offer adequate throughput, optimizing *throughput stability*, a critical determinant of rebuffer rate performance, emerges as essential for short video services. Beyond traditional network metrics such as RTT, loss, and bandwidth, our research also highlights the significant impact of network operators' traffic policies, such as rate limiting (refer to Observation III). Taking these policies into consideration is the key to optimal CC algorithm design. Furthermore, CCs customized for specific network types such as Wi-Fi and 4G, considering their unique physical characteristics, greatly enhance short video QoE (refer to Observation II).

Data cleaning. The massive number of CDN nodes requires careful data pre-processing via automatic data cleaning tools, augmented with our expertise in the patterns of TCP statistics and the prediction target, *i.e.,* network types. In our experience, data like RTT often incurs long-tail outliers that are extremely high but occur infrequently, thus violating the Gaussian assumption explicitly assumed by most deep-learning approaches, and impacting the efficiency of data normalization. Some abnormal RTT values can even skew the average RTT to seconds. Our current solution is to compress the data range by applying a logarithmic transformation (log), followed by standardization. This approach avoids the need for manually deciding the threshold to filter out outliers and significantly aids in accelerating model training.

Advantages of distributed inference for CDN networks. Our application of ML to CDNs indicates a distributed inference architecture is more beneficial for CDN networks. This approach offers two benefits: First, it enhances scalability by enabling edge nodes to independently serve requests, which alleviates bottlenecks and increases the network's capacity to manage high traffic volumes. Second, since CDN central servers often reserve processing power for video transcoding, offloading inference tasks to edges with surplus CPU cycles for peak demand is more cost-effective. In contrast, centralizing these tasks requires allocating an additional 2 CPU cores and 4 GB of memory to each of thousands of CDN nodes, resulting in significant expenses for CDN based on tests.

Enhanced gains of customized network services for IPv6. Our deployment experience shows that customizing network services using IP prefix aggregation reduces average rebuffer rates by 60% more in IPv6 networks than in IPv4 (7.6% vs 4.76%). This arises from two factors: First, IPv6 enhances predictability by reducing reliance on NAT and similar technologies used in IPv4. IPv6 network tends to assign a fixed IP prefix to a subnet over time, improving history-based predictions. Conversely, IPv4 with NAT may frequently change the mapping of IP prefix to underlying subnet, undermining these predictions. Second, the disparity in network characteristics between Wi-Fi and 4G is more pronounced in IPv6, with a ratio between mean retransmission rate of Wi-Fi over 4G being 2.1× in IPv4 and 2.9× in IPv6. These facts underscore the need for more tailored services to manage Wi-Fi and 4G disparities in IPv6 networks.

Deploying a fallback strategy for ML in CDN networks. When integrating ML system with CDNs, it is crucial to establish a robust fallback strategy when ML predictions are inaccurate. In practice, no ML system can perfectly address every scenario or maintain reliability in all situations. Therefore, it's essential to revert to a default strategy when ML predictions are uncertain. We employ a threshold-based scheme: if our ML-predicted network type has a logit exceeding 80% for Wi-Fi type and 60% for 4G type, it is trusted; otherwise, the system defaults to its original protocol set-up. The higher threshold for Wi-Fi reflects more cautious decision-making due to its large request proportions compared to 4G. This safeguard ensures that performance does not drop below the default configuration, thus enhancing reliability.

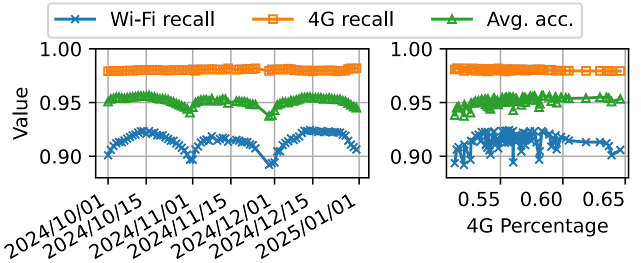


Figure 9: Model prediction accuracy varying date (left) and percentage of 4G traffic (right).

5 Evaluation

To ensure a smooth transition to full-scale deployment, we have focused our efforts on short video services. AliCCS has been applied to two major short video services on Alibaba Cloud CDN, with one of them contributing training data over 30% of our CDN nodes. We evaluate AliCCS's performance by addressing three key questions:

Q1: How accurate is our ML model in predicting the network types for large-scale, geo-distributed requests?

Q2: How efficient is our inference pipeline, particularly with the cache-based, online-offline decoupled design?

Q3: What's the QoE improvement for short video services using our production CDN?

5.1 Machine Learning Model Accuracy

Model prediction effectiveness. We validate our model's prediction accuracy across numerous CDN nodes handling high volumes of short video requests from major ISPs in China. To ensure statistical reliability, nodes with less than 5% of requests from 4G or Wi-Fi connections are excluded. Our ML model achieves an accuracy ranging from 95.8% to 99.0%, averaging an improvement of 1.58% and peaking at 3.4% on Wi-Fi networks of ISP #2. Remarkably, this is accomplished using only 30% CDN nodes for model training, confirming our model's generalizability.

Online validation of the ML model. Figure 9 presents our model's prediction accuracy in serving a large volume of short video requests over 3 months on a CDN node. The left figure depicts that our ML model achieves a remarkably high value of recall (*i.e.,* true positive rate of predicting each network type), varying slightly between 89% to 93% for Wi-Fi, 97% to 98% for 4G, and an overall average ranging from 94% to 96%. Interestingly, the Wi-Fi recall exhibits a monthly cyclic pattern and remains stable over time. This phenomenon likely stems from ISP-enforced monthly 4G data caps, leading to a reduction in 4G usage as users hit their data limits towards the end of the month and a corresponding drop in Wi-Fi recall at the same times. This dependency is due to our model being trained on a balanced dataset (*i.e.,* 50% of the data is 4G), not reflecting the actual distribution of 4G usage at the beginning and end of the month. Previous studies, such as in reference [39], have shown that when the proportions of category (like 4G) shift, classification accuracies decrease. The impact on Wi-Fi recall is more noticeable due to its smaller absolute number. Additionally, our model prediction accuracies for both Wi-Fi and 4G are robust to varying percentages of 4G connections, as demonstrated in Figure 9 (right), confirming ALICCS's generalizability. We continuously monitored the model's performance by checking a domain name whose requests have been labeled by App #1 short video service provider. We see that the model maintains a high recall rate of over 90% for each network type for at least 6 months, from March 2024 to August 2024.

ML model prediction comparison. To evaluate the effectiveness of our ML model design, we conduct ablation studies, evaluating ALICCS's different components, and compare with Configanator, an advanced learning-based CCS scheme, and a baseline named Vanilla ML, where we directly feed training data into an MLP (multi-layer perceptron) network without extra generalization optimization. As shown in Figure 10, Configanator shows similar average performance as Vanilla ML but exhibits significantly more performance variance. This is due to Configanator's method of aggregating samples by network class, resulting in overfitting when the number of samples is small. Our GAN-based design significantly reduces variance across CDN nodes, an essential feature for large-scale deployments that demand reliability in worst-case scenarios. This is potentially due to our tailored regularizers in the loss functions, which leverage domain knowledge to boost model robustness by promoting learning from a universal set of features. For instance, incorporating RSC improves average performance by encouraging diverse feature use, and our regularizer $L_{variance}$ reduces feature noises, thereby improving average prediction accuracies.

5.2 Implementation Framework Efficiency

Superiority of our offline-online decoupled inference. In Table 3, we compare our decoupled online-offline inference method with the baseline, which executes *serial per-request online inference*. In the experiment, we set 32 Nginx workers to fully utilize available CPUs of 32 cores on the machine, emulating a CDN node at maximum QPS. The baseline method incurs a 10417ns processing delay and handles a maximum of 7.6k QPS only, while our system significantly reduces the processing delay to 162 ns and achieves 18.4k maximum QPS, representing 64.30× and 2.42× improvements respectively. Consequently, our decoupled framework drastically cuts AI inference delays and substantially increases QPS capacity, which is critical for managing high user demand efficiently.

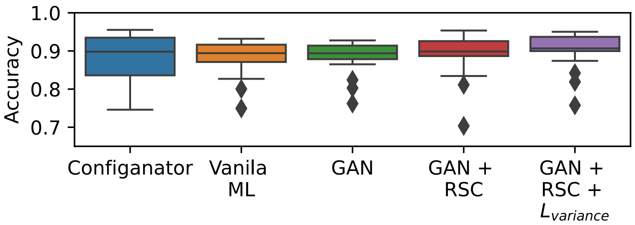


Figure 10: Ablation studies for evaluating different components of ALICCS and comparison with Configanator [32].

Table 3: Comparison with non-optimized online inference.

	Processing delay	Max. QPS
Baseline: online inference	10417 ns	7.6k
Online-offline decoupled	162 ns	18.4k
Improvement	64.30×	2.42×

CPU and Memory usage patterns. As Shown in Figure 11, the CPU overhead of ALICCS grows slowly with the number of connection requests, reaching a maximum of 200% at 17k QPS (refer to the left figure). This indicates our system requires no more than 2 cores, which is less than 1% (2 out of 256 cores) of the total CPU resources per CDN node, to handle peak traffic. Additionally, ALICCS's memory usage remains conservative, never exceeding 2.9GB (refer to the right figure), over a seven-day period in our measurements. These results underscore ALICCS's minimal resource impact on the overall CDN resource availability and its significant run-time efficiency. Additionally, our results also reveal interesting usage patterns in short video service: People tend to watch more short videos at noon, evidenced by the CPU usage being proportional to request volume and the 2 minor daily peak patterns; After a slight decrease early in the afternoon, viewship steadily climbs until reaching the peak at 20 o'clock.

5.3 QoE and Retransmission Improvement

In this section, we present the improvement in both QoE (rebuffer rate) and bandwidth cost (retransmission rate) achieved by ALICCS, serving the two major short video services, App #1 and App #2, in our Alibaba Cloud CDN.

Evaluation setup. To demonstrate the QoE improvements from deploying ALICCS, we conduct a month-long Randomized Control Test (RCT), comparing rebuffer rates with and without CCS. To approximate the random assignment of client connections to both versions, for each connection request, we apply the Cyclic Redundancy Check (CRC) [36] hash function on the client's IP address. Depending on whether the hash code is odd or even, the request is assigned to the system with our CCS strategy or the default system without CCS.

Baseline. The baseline system (default) uses CUBIC for all

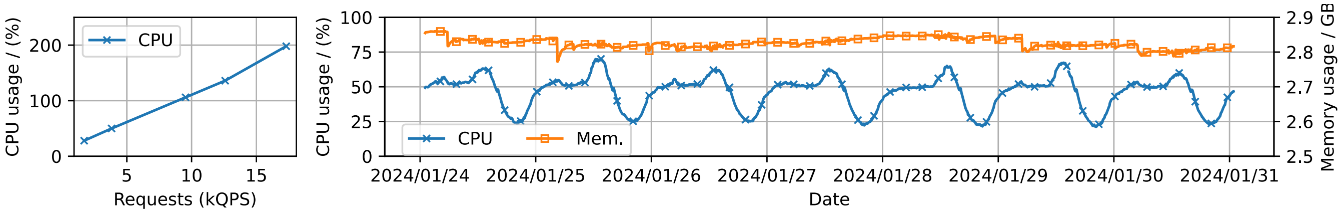


Figure 11: ALICCS's CPU overhead varying kQPS (left) and its seven-day usages of CPU and memory (right).

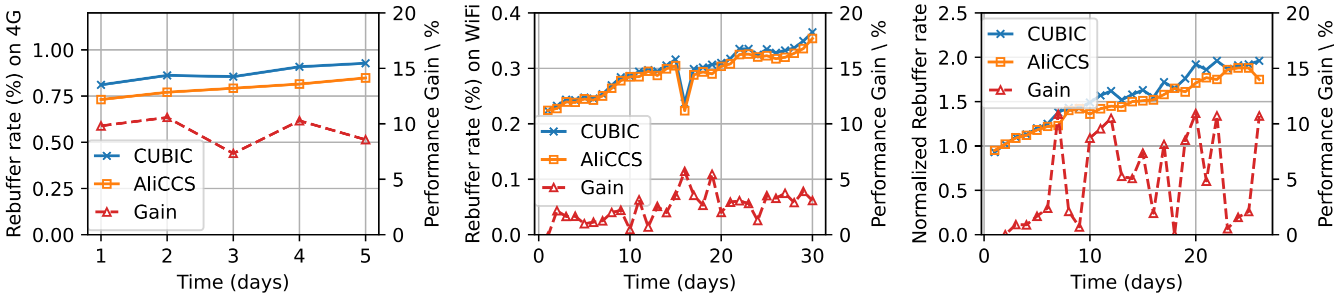


Figure 12: End-to-end rebuffer rate comparison of CUBIC and ALICCS.

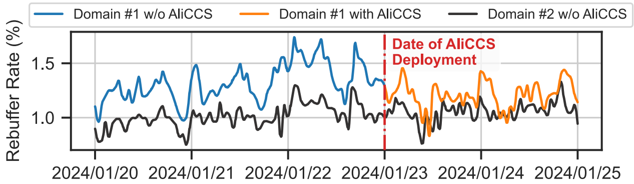


Figure 13: Rebuffer rates over time for two domain names with the same CDN nodes. The blue and orange curves represent the rebuffer rate for Domain #1 before and after ALICCS was applied. The black curve, associated with Domain #2, confirms stable CDN network conditions in this period of time.

requests given Wi-Fi's dominance in our CDN connections.

Real-world rebuffer rate improvement. As shown in Figure 12, ALICCS consistently achieves a lower end-to-end rebuffer rate, with improvements of 9.31% for 4G and 2.51% for Wi-Fi, along with a 4.76% gain across all connections *w.r.t.* deployment prior to ALICCS with all requests using CUBIC. This 5% reduction can boost our market edge in the fiercely competitive CDN industry, where even a 2%–3% difference in service quality matters, ensuring customer retention. Figure 13 zooms in on the results around the time when ALICCS was deployed, demonstrating the rebuffer rate reduction is attributed to the deployment of ALICCS.

Real-world retransmission rate improvement. As demonstrated in Figure 14, ALICCS achieves 59.24% improvement on App #1 and 61.28% on App #2, which translates into tens of million US dollars savings per year in terms of bandwidth cost for Alibaba Cloud CDN. To better understand the geographical diversity, we also investigated the performance gains across each province in China, as shown on the right side of Figure 14. ALICCS consistently achieves significantly lower retransmission rate in all provinces, with minimum, maximum, average, and standard deviation gains of 25.51%, 174.36%, 62.76%, and 36.28%, respectively. The results also lead to several interesting findings: Henan province tops the chart, possibly due to its high population density and heavy reliance on 4G networks; Performance gains in Guangdong province and Northeastern China exceed the average, in line with our observed widespread habit of streaming short videos on mobile devices in these areas. The smallest gain is seen in Qinghai province, likely attributed to inadequate 4G connectivity and lower population density.

5.4 Trace-driven Evaluation

Emulation setup. We also conduct an emulation study using short video traces collected over a day from one of Alibaba Cloud's highest-traffic CDN nodes. We compare ALICCS with two model-based CCS methods, Configanator [32] and Disco [47], both of which utilize the decision tree, as well as Pytheas [26], an online learning-based CCS (The rationale for the baseline selections is detailed in Appendix §E). We select CUBIC and BBR as CC candidates for evaluation due to their extensive testing, deployment, and worst-case assurance. Additionally, we construct the *best CCS* based on our trace, named Oracle, as a benchmark for CCS algorithm comparisons. We also define an aggregate QoE function, defined as a linear combination of rebuffer and retransmission rates, *i.e.,* $QoE = 1 - \alpha \cdot rebuf - (1-\alpha) \cdot retrans$. Given our primary focus is on rebuffer rate over transmission costs, we set $\alpha = 0.8$ to prioritize minimizing stalls during short video streaming. Our emulation study employs a time-window-based mechanism [25, 27], where we construct unbiased estimates of CC's *outcomes*, *i.e., QoE*, rebuffer rate, and retransmission rate metrics. For each /24 IP prefix group, we assign a time window for each CC to collect its corresponding outcomes. Oracle selects the CC with the best average *QoE* at the end of each window for each /24 IP prefix group. To simulate any

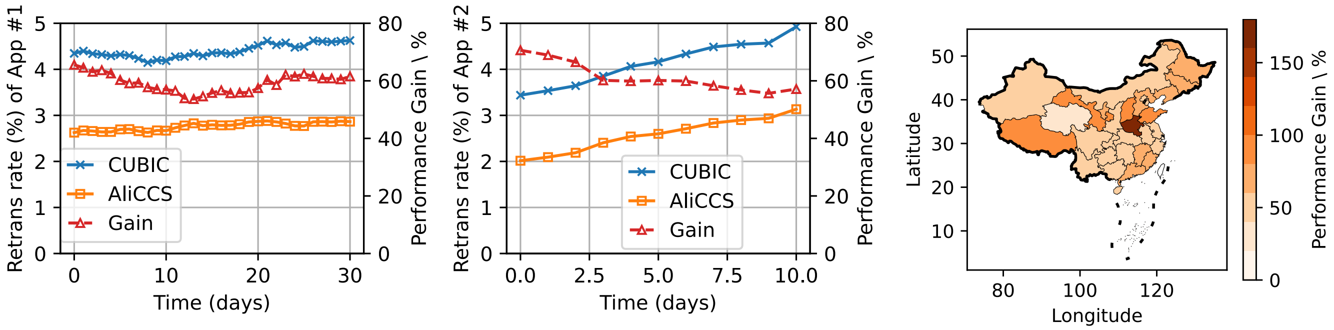
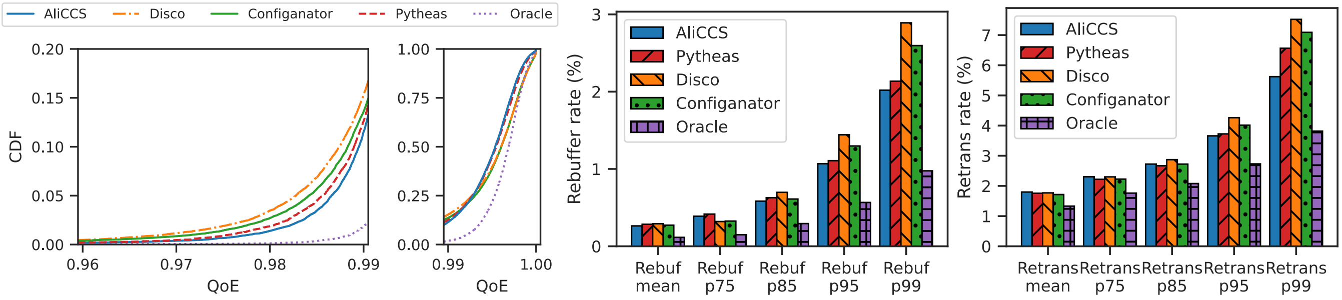


Figure 14: End-to-end retransmission rate comparison of CUBIC and AliCCS.



(a) QoE in good (right) and poor (left) network conditions.

(b) Rebuffer rate.

(c) Retransmission rate.

Figure 15: End-to-end improvement achieved by AliCCS in the trace-driven emulation.

algorithm's outcomes for CCs, including those that are not actually adopted by the real-world policy at a given timestamp in our trace, we use the average outcomes at historical times where the same CC is chosen under the same IP prefix group and network type.

Comparison with CCS state-of-the-arts. As shown in Figure 15a, all algorithms perform similarly in high QoE regions (right), *i.e.,* good network conditions where CC choice has minimal impact. In contrast, AliCCS performs notably well in low QoE regions (left), *i.e.,* impaired conditions. For QoE metrics like rebuffer rate, samples from poor network conditions disproportionately affect the average, underscoring AliCCS's advantage in low QoE regions. We then compare baseline performance under poor network conditions. Figure 15b shows that AliCCS *achieves a 4.54% to 30.80% lower rebuffer rate in the 85th percentile and beyond.* In contrast, Configanator and Disco exhibit higher rebuffer rates, as they overfit to good network conditions, neglecting the importance of edge cases with poor network scenarios. Meanwhile, Pytheas performs sub-optimally due to the inherent exploration required during network probing. Figure 15c shows the retransmission rate results. While AliCCS slightly increases retransmissions to reduce rebuffering, it outperforms Configanator, Disco, and Pytheas in the long-tail distribution, achieving a 1.74% to 25.24% lower retransmission rate. While Pytheas achieves comparable performance to AliCCS, estimating its reward function (*e.g.,* rebuffer rates) in real-time poses significant challenges and could incur additional computational overhead compared to AliCCS.

6 Related Work

Congestion control (CC) is a longstanding and significant topic, featuring a multitude of meticulously designed CCs, including CUBIC [20], BBR [12], and Reno [23].

Automatic CC Configuration. Recent studies propose machine-generated CCs and automatic parameter adjustments, highlighting that handwritten CC policies are effective only in limited contexts. Remy [45] utilizes offline optimization to generate a new CC by searching from a certain design space, followed by Copa [9] with more theoretical insights. CherryPick [8] employs Bayesian Optimization, while CFA [25] exploits domain-specific insights. Such methods fail to leverage large-scale data available on CDNs, potentially limiting their accuracy and generalization performance.

Learning-based CC parameter tuning. To increase generalizability, ML has been adopted to adjust CC parameters. For instance, Orca [6], Aurora [24], Libra [15], and TCP-RL [33] adopts RL with handcrafted reward functions for learning parameters like congestion windows, sending rates, etc. A few others adopt online learning but generally favor scenarios with i.i.d. network and/or CC characteristics. For example, Pytheas [26] applies multi-armed bandits (MAB), Configanator [32] integrates MAB with decision trees, and PCC vivace [14] utilizes gradient-based online learning. Recent works also adopt imitation learning, *e.g.,* DuGu [21] and Muses [51]. Such methods often overlook crucial worst-case scenarios in production CDN. AliCCS, instead, uses GAN-based model to balance performance across scenarios and tackles the often-overlooked inference latency.

Learning-based CCS. Recent works have advocated CCS, *i.e.,* selecting the best CC among existing schemes. Specifically, Rein [13] uses data-driven methods to mitigate the CC transitioning overhead. Disco [47] and Antelope [52] leverage RL-inspired reward functions, demonstrating the advantage of CCS over CC tuning. OPSBC [50] combines traditional flow scheduling methods with MAB-based CCS. Sage [49] employs offline DRL to learn a new CC from traces, despite the complexity in feedback collection and high training cost due to a large state-action space. DPO [1] from Akamai represents an industrial effort to optimize CCS. Unlike ALICCS, these works do not present large-scale network experimental results, nor do they incorporate CDN domain knowledge.

Domain adaptation aims to preserve model performance on a target domain with limited data by applying knowledge from a source domain, and has been applied in various networking problems. For instance, Genet [46] applies curriculum learning for network adaptation. AWARE [37] uses meta-learning and bootstrapping for cloud auto-scaling. Other works leverage knowledge from simulation settings [41] or smaller-scale networks [44]. These works shed light on our work, albeit from a high-level perspective. We instead adopt GAN and Causal Graphical Model (CGM), tailoring our design specifically for short video workloads and production CDNs. Our solution is deeply rooted in CDN-specific domain knowledge.

7 Discussion

Do the observations driving ALICCS' design remain valuable beyond the scope of short video services? We believe most of our measurement insights extend beyond short videos and can be significant for the broader community. Notably, Observations I, III, and IV in §2.3—highlighting the variability in CCs' advantages, the impact of rate limiting, and the dynamics of network types—apply in general and inform further research on intelligent CCS design for diverse services. Additionally, our lessons in §4.2 on data cleaning, distributed inference, IPv6 benefits, and the fallback strategy provide insights for designing ML-based CDN systems. However, Observation II on correlation between network types and the best CCS and the lesson on throughput stability, are more specific to short video contexts, which is likely tied to the low traffic volume of short videos and the dependency on rebuffer rate for QoE.

Can ALICCS' design be effectively applied outside of short video domain or Alibaba Cloud's CDN? Our GAN-based model (§3.1), which learns uniform feature representations across IP prefixes, can be applied to CCS prediction for any service, provided that there is sufficient A/B test data with relevant labels (*i.e.,* best CCS under an IP prefix). The interpretation technique (§3.2) can aid model debugging of any classification task. Moreover, the distributed inference pipeline (§3.3) should be widely applicable to many ML-based CDN systems to reduce latency with a few assumptions: the input/output features of ML model are temporally stable and similar under the IP prefix, which are commonly adopted in existing works [26, 32]. However, it is noteworthy that, as rate limiting affects CCS (§2.3), ISP policy differences may impact ALICCS' performance gains in other networks.

Does network contention among clients degrade ALICCS' prediction accuracy? ALICCS uses the GAN model to achieve balanced accuracy across diverse conditions, including contention. In fact, ALICCS outperforms existing works by reducing accuracy variance and 95th percentile rebuffer rate, likely associated with impaired conditions (see Figures 10 and 15). However, ALICCS does not strictly optimize accuracy uniformity, as we find this could degrade overall accuracy significantly. Instead, we fine-tune λ_1 in the training loss to balance accuracy uniformity and overall accuracy.

ALICCS's reliance on pre-deployment test and domain knowledge: The success of framing CCS as an ML classification task relies heavily on comprehensive pre-deployment tests and domain knowledge to identify a strong link between network types and best CCS. This framework eliminates the need for acquiring short video QoE training data directly from content providers. However, it is not always straightforward, for a new scenario, to establish a clear connection between model predictions and desired outcomes. Our approach, despite avoiding continuous observations and handcrafted reward functions like in RL approaches, demands deep domain knowledge and insightful observations.

8 Conclusion

In this paper, we introduce ALICCS, the first production-optimized CCS approach that has served Alibaba Cloud CDN for over one year. ALICCS exploits our insights from large-scale measurements that reveal contrasting CC performances for short video services in varying network access types. It employs a domain-specific ML model for scalability and generalizability across numerous CDN nodes and changing network conditions, with an efficient inference pipeline for runtime performance. Extensive results from our production CDN demonstrate that ALICCS achieves significant retransmission rate reduction and enhanced QoE for short video services, outperforming advanced ML approaches and the default rule-based CC. We plan to explore combining CCS with protocol parameter tuning in the future and assess whether this integration can yield further QoE improvements.

Acknowledgments

We thank our shepherd, Soheil Abbasloo, and the anonymous reviewers for their insightful comments. This work is supported by NSFC under grant 62472460, Guangdong Basic and Applied Basic Research Foundation under grants 2024A1515010161 and 2023A1515012982, Young Outstanding Award under the Zhujiang Talent Plan of Guangdong Province, and the Alibaba Research Intern Program. Xiaoxi Zhang and Ennan Zhai are the co-corresponding authors.

References

[1] DPO from Akamai. https://www.linkedin.com/pulse/need-dynamic-protocol-optimization-ant-bao, 2020.

[2] MaxMind GeoIP database. https://www.maxmind.com/en/home, 2020.

[3] information Gain from wiki. https://en.wikipedia.org/wiki/Information_gain_(decision_tree), 2024.

[4] The Evolution of CDN Technology in 2024. https://blog.blazingcdn.com/en-us/cdn/the-evolution-of-cdn-technology-in-2024, 2024.

[5] What are QUIC and HTTP/3? https://www.f5.com/glossary/quic-http3, 2024.

[6] 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.

[7] 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 *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[8] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[9] Venkat Arun and Hari Balakrishnan. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, 2018.

[10] Hirochika Asai and Yasuhiro Ohara. Poptrie: A compressed trie with population count for fast and scalable software ip routing table lookup. *ACM SIGCOMM Computer Communication Review*, 45(4):57–70, 2015.

[11] Lawrence S Brakmo, Sean W O'malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, 1994.

[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] 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.

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[15] 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.

[16] Sishuai Gong, Usama Naseer, and Theophilus A Benson. Inspector gadget: A framework for inferring tcp congestion control algorithms and protocol configurations. In *Network Traffic Measurement and Analysis Conference*, 2020.

[17] Ali Gouta, David Hausheer, Anne-Marie Kermarrec, Christian Koch, Yannick Lelouedec, and Julius Rückert. Cpsys: A system for mobile video prefetching. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 188–197. IEEE, 2015.

[18] Carlo Augusto Grazia, Natale Patriciello, Toke Høiland-Jørgensen, Martin Klapez, and Maurizio Casoni. Aggregating without bloating: Hard times for tcp on wi-fi. *IEEE/ACM Transactions on Networking*, 30(5):2359–2373, 2022.

[19] Pankaj Gupta and Nick McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[20] 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.

[21] Tianchi Huang, Chao Zhou, Lianchen Jia, Rui-Xiao Zhang, and Lifeng Sun. Learned internet congestion control for short video uploading. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 3064–3075, 2022.

[22] Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, proceedings, part II 16*, pages 124–140. Springer, 2020.

[23] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

[24] 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.

[25] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. {CFA}: A practical prediction system for video qoe optimization. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 137–150, 2016.

[26] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *NSDI*, volume 1, page 3, 2017.

[27] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[28] Stan Lipovetsky and Michael Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

[29] Ge Ma, Zhi Wang, Minghua Chen, and Wenwu Zhu. Aprank: Joint mobility and preference-based mobile video prefetching. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 7–12. IEEE, 2017.

[30] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.

[31] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[32] Usama Naseer and Theophilus A Benson. Configanator: A data-driven approach to improving cdn performance. In *19th USENIX NSDI Symposium*, 2022.

[33] 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.

[34] Stefan Nilsson and Gunnar Karlsson. Ip-address lookup using lc-tries. *IEEE Journal on selected Areas in Communications*, 17(6):1083–1092, 1999.

[35] J Pearl. Causality: Models, reasoning, and inference 47cambridge university presscambridge, united kingdom. pearl, j. 2000. *Causality: models, reasoning, and inference*, 47, 2000.

[36] William Wesley Peterson and Daniel T Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[37] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. {AWARE}: Automate workload autoscaling with reinforcement learning in production cloud systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 387–402, 2023.

[38] Miguel Á Ruiz-Sánchez, Ernst W Biersack, and Walid Dabbous. Survey and taxonomy of ip address lookup algorithms. *IEEE network*, 15(2):8–23, 2001.

[39] Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural computation*, 14(1):21–41, 2002.

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

[41] Junyang Shi, Mo Sha, and Xi Peng. Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 887–901, 2021.

[42] Tanya Shreedhar, Rohit Panda, Sergey Podanev, and Vaibhav Bajpai. Evaluating quic performance over web, cloud storage, and video workloads. *IEEE Transactions on Network and Service Management*, 19(2):1366–1381, 2021.

[43] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.

[44] Shun Tobiyama, Bo Hu, Kazunori Kamiya, and Kenji Takahashi. Large-scale network-traffic-identification method with domain adaptation. In *Companion Proceedings of the Web Conference 2020*, pages 109–110, 2020.

[45] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review*, 43(4):123–134, 2013.

[46] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[47] Furong Yang, Zhenyu Li, Jianer Zhou, Xinyi Zhang, Qinghua Wu, Giovanni Pau, and Gaogang Xie. Disco: A framework for dynamic selection of multipath congestion control algorithms. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2023.

[48] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X Liu, Qi Li, and Laurent Mathy. Guarantee ip lookup performance with fib explosion. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 39–50, 2014.

[49] Chen-Yu Yen, Soheil Abbasloo, and H Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 255–274, 2023.

[50] Xiaoxi Zhang, Siqi Chen, Yunfan Zhang, Youngbin Im, Maria Gorlatova, Sangtae Ha, and Carlee Joe-Wong. Optimal network protocol selection for competing flows via online learning. *IEEE Transactions on Mobile Computing*, 22(8):4822–4836, 2023.

[51] Zhiren Zhong, Wei Wang, Yiyang Shao, Zhenyu Li, Heng Pan, Hongtao Guan, Gareth Tyson, Gaogang Xie, and Kai Zheng. Muses: Enabling lightweight learning-based congestion control for mobile devices. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2208–2217. IEEE, 2022.

[52] 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.

Appendices

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

A Methodology to Conduct A/B Test Comparing CUBIC and BBR

To conduct a fair A/B test to compare performance of CUBIC and BBR in short video services, for each new connection from a specific partner app, we hash the request's IP using CRC algorithm to randomly map it to either the CUBIC or BBR group, ensuring a fair workload and network environment. The measurement period lasted around two weeks, with average data collected on a regional and daily basis. We have checked and confirmed the consistency of the results over this period. To further valid the difference we observe is due to CC's assignment rather than inherent difference between the group, we have also reversed the A/B assignment (IP assigned to group A now assigned to group B and vice versa) for about another two weeks, and the results are consistent with the first test, which validates the fairness of A/B test and the difference we observed is due to CC's assignment.

B Input Features of Our ML Model

Our CDN network collects several dozens of metrics to support a wide range of applications and services. Among these metrics, we carefully pick twelve of them as input features for our ML model, as shown in Table 4.

Table 4: Input Features for the ML Model

Feature	Description
mss	TCP maximum segment size
connection_time	TCP handshake time
init_snd_wnd	receiver initially advertised TCP window
min_rtt	minimum RTT in a TCP connection
max_snd_wnd	TCP receiver advertised maximum window size
min_snd_wnd	TCP receiver advertised minimum window size
min_rtt_variance	min RTT variance
max_rtt_variance	max RTT variance
min_rto	minimum retransmission timeout
min_srtt	max smoothed version of RTT
max_srtt	max smoothed version of RTT
max_cwnd	maximum congestion window of a TCP sender

C Handle Non-1:1 Mapping between Connections and Network Types

When there is a connection migration from one network type to another, there could be non-1:1 mapping between Connections and Network Types. We dicuss potential solutions to it. In case the migration is transparent to CDN server (e.g., the migration is behind a mobile proxy server), we find changing the CC for a live connection is a heavy engineering task requiring significant kernel modifications, and such cases are in minority. So we decide they will not be handled by AliCCS. In case the migration is visible to CDN server, like in case where QUIC or MPTCP is applied, AliCCS can be applied independently for each sub-connection, leveraging two IPs to find the best CC for each sub-connection in its cache.

D Hyper Parameter Tuning and Experiences

We report below some details on the engineering decisions and parameter choices. We explain as well the key considerations and the rationale behind our decisions.

Hyper parameters in the loss function for DL model training. We aim to strike a balance between reducing the variability of prediction performance across CDN nodes and maximizing overall average prediction accuracy. Our experiments show that increasing the parameters λ_1, λ_2 encourages the model to extract more uniform representations, thereby reducing the variability of prediction accuracy across CDN nodes. However, strictly minimizing variability could hurt the over all average accuracy. To address this, we perform a grid search to determine the values of λ_1 and λ_2, achieving minimal variability of accuracy without significantly compromising overall accuracy.

Other hyperparameters in DL model design. While the best hyperparameter configurations provided in existing literature on GANs can serve as a a starting point, we observe that they do not deliver the expected performance boost in our scenario. This likely reflects domain differences between areas like computer vision and networking. Consequently, we rely on performing a grid search to obtain the best-performing hyperparameter set for AliCCS, tailored to its unique requirements.

Inference pipeline at CDN Nodes. To balance computational efficiency and prediction accuracy, we opted to decide update the cache on an hourly basis. For our IP prefix aggregation strategy, we employ differentiated thresholds: For IP prefixes with high request volumes, we set high thresholds to make the conditions more strict, in order to avoid incorrect predictions affecting a large number of requests. For IP prefixes with low request volumes, we apply low thresholds to encourage aggregation, reducing memory usage without significantly impacting prediction accuracy.

E Method Selection Rationale for Trace-driven Evaluation

In our trace-driven evaluation, several state-of-the-art methods were not selected for the following reasons. Adapting Sage [49], originally designed for new CC learning to a CCS scheme, presents challenges due to its inefficiencies for production CDNs. The impracticality stems from the extensive training data required, such as rebuffer rates and bandwidth costs, across various state configurations from short video providers, which is not applicable for us as a CDN provider.

Algorithm 1 Prediction Results Aggregation Algorithm

```
**Input**: Pairs of IP and predicted cc: ${(IP_i, pred\_res_i)}_{i=1}^m$
**Output**: $hashtable : IP\_prefix \to (pred\_res, purity)$
**Input Parameters**: $p\_thresh\_16, p\_thresh\_24,$
// step1: aggregate at prefix /16
Initialize $D \leftarrow \{(IP_i, pred\_res_i)\}_{i=1}^m$
$H \leftarrow$ AGGREGATE_AT_PREFIX_LEN$(D, 16)$
// step2: for prefix without sufficient purity, expand at prefix /24
$D\_expand\_24 \leftarrow \emptyset$
for all prefix_16 in $H.keys()$ do
    if $H[prefix\_16].purity < p\_thresh\_16$ then
        $D\_tmp \leftarrow$ { $IP_i, pred\_res_i$ | $IP_i$ matches prefix_16 }
        $D\_expand\_24 \leftarrow D\_expand\_24 \cup D\_tmp$
        remove prefix_16 from $H$
    end if
end for
// step3: for prefix without sufficient purity, expand at prefix /32
$H\_24 \leftarrow$ AGGREGATE_AT_PREFIX_LEN$(D\_expand\_24, 24)$
$D\_expand\_32 \leftarrow \emptyset$
for all prefix_24 in $H\_24.keys()$ do
    if $H\_24[prefix\_24].purity < purity\_thresh\_24$ then
        $D\_tmp \leftarrow$ (find all data under prefix_24)
        $D\_expand\_32 \leftarrow D\_expand\_32 \cup D\_tmp$
        Delete prefix_24 from $H\_24$
    end if
end for
$H\_32 \leftarrow$ AGGREGATE_AT_PREFIX_LEN$(D\_expand\_32, 32)$
// merge the 3 hash tables
$H \leftarrow H \cup H\_24 \cup H\_32$
return $H$

procedure AGGREGATE_AT_PREFIX_LEN($D$, $prefix\_len$)
    Input: $D$: a subset of Pairs of IP and prediction result
                $prefix\_len$: the length of prefix to aggregate at
    Output: $Map : IP\_prefix \to (pred\_res, purity)$
    Initialize $H \leftarrow \emptyset$                    ▷ a 2-level hashtable
    for all $(IP, pred)$ in $D$ do
        $ip\_prefix \leftarrow$ COMPUTE_PREFIX$(IP, prefix\_len)$
        $H[ip\_prefix][pred] \leftarrow H[ip\_prefix][pred] + 1$
    end for
    Initialize $H\_res \leftarrow \emptyset$
    for all ip_prefix in $H.keys()$ do
        $pred\_label \leftarrow \arg\max_{pred}(H[ip\_prefix][pred])$
        $purity \leftarrow (fractionofpred\_labelunderip\_prefix)$
        $H\_res[ip\_prefix].pred\_res \leftarrow pred\_res$
        $H\_res[ip\_prefix].purity \leftarrow purity$
    end for
    return $H\_res$
end procedure
```

Both Disco [47] and Antelope [52] employ tree-based ML models for CCS prediction, with Antelope permitting mid-flow CC switching. However, in our trace-driven emulation where CCS is allowed only on a per-flow basis, the two approaches become nearly identical. Therefore, we implement only Disco as a representative model. Rein [13] was excluded from our comparative study due to its closed-source nature and its heuristic rules not being tailored for our short video service context.

F Prefix Aggregation Algorithm

We illustrate the prefix aggregation algorithm with a flow chart shown in Figure 16 as well as with Algorithm 1. Initially, we aggregate IP prefixes at the /16 level using a Trie and then assess the purity of this aggregation. Purity is defined as the ratio of the number of IP addresses with the most common label to the total number of IP addresses within the range. A high purity indicates that using the most common label to represent all IPs within the range results in minimal accuracy loss. If the purity is insufficient, we further refine the aggregation to /24 and /32 prefixes, as detailed in Steps 2 and 3 of the algorithm.

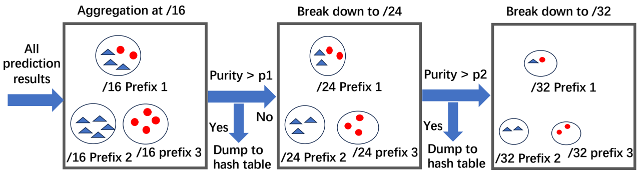
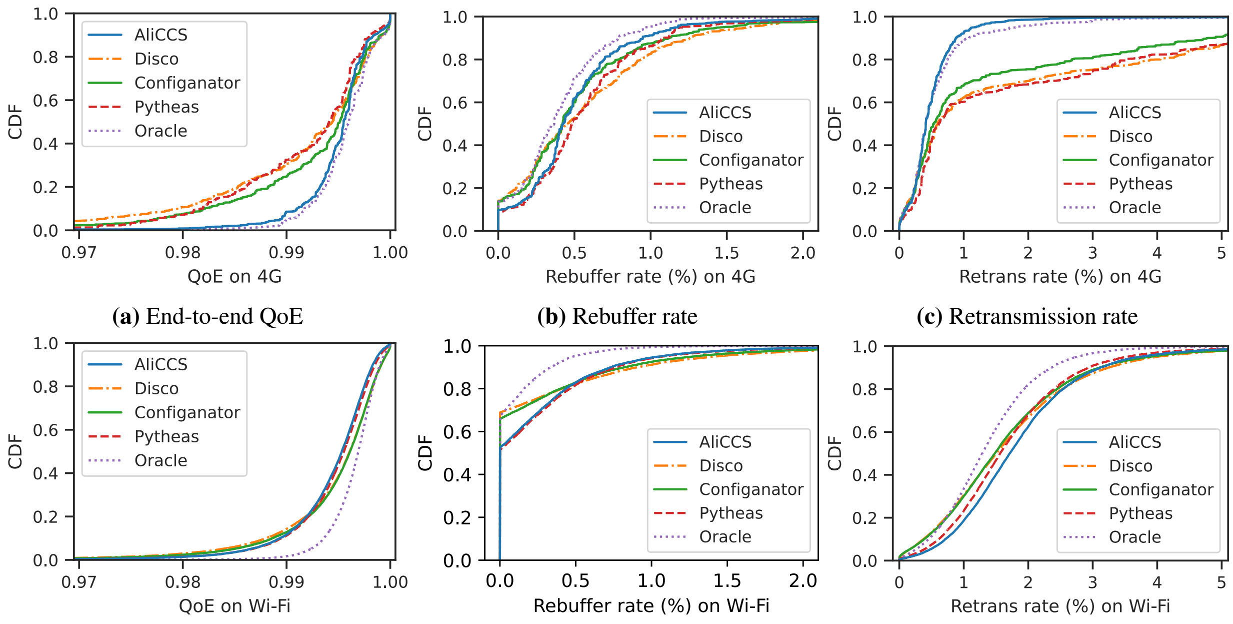


Figure 16: Trie-based prefix aggregation algorithm.

G End-to-End Performance

Figure 17 shows the end-to-end performance comparison of WiFi connections and 4G connections in terms of QoE, rebuffer rate, and retransmission rate.



(a) End-to-end QoE

(b) Rebuffer rate

(c) Retransmission rate

(d) End-to-end QoE

(e) Rebuffer rate

(f) Retransmission rate

Figure 17: The enhancement of ALICCS in trace-driven emulation is shown in terms of QoE, rebuffer rate, and retransmission rate. Figures 17a–17c show the results for 4G connections, while Figures 17d–17f show the results for Wi-Fi connections.