Learning to Coordinate Video Codec with Transport Protocol for Mobile Video Telephony

Anfu Zhou[+], Huanhuan Zhang[+], Guangyuan Su[+], Leilei Wu[+], Ruoxuan Ma[+], Zhen Meng[+], Xinyu Zhang[◇], Xiufeng Xie[*], Huadong Ma[+], Xiaojiang Chen[†*]

[+]Beijing Univ. of Posts and Telecom. {zhouanfu, zhanghuanhuan, mengzhen,mhd}@bupt.edu.cn

[◇] University of California San Diego xyzhang@ucsd.edu

[*] Hewlett Packard Labs xiufeng.xie@hpe.com

[†] Alibaba Inc. zhongsheng.cxj@taobao.com

ABSTRACT

Despite the pervasive use of real-time video telephony services, the users' quality of experience (QoE) remains unsatisfactory, especially over the mobile Internet. Previous work studied the problem via controlled experiments, while a systematic and in-depth investigation in the wild is still missing. To bridge the gap, we conduct a large-scale measurement campaign on Taobao-Live, an operational mobile video telephony service. Our measurement logs fine-grained performance metrics over 1 million video call sessions. Our analysis shows that the application-layer video codec and transport-layer protocols remain highly uncoordinated, which represents one major reason for the low QoE. We thus propose Concerto, a machine learning based framework to resolve the issue. Instead of blindly following the transport layer's estimation of network capacity, Concerto reviews historical logs of both layers, and extracts high-level features of codec/network dynamics, based on which it determines the highest bitrates for forthcoming video frames without incurring congestion. To attain the ability, we train Concerto with the aforementioned massive data traces using a custom-designed imitation learning algorithm, which enables Concerto to learn from past experience. We have implemented and incorporated Concerto into Taobao-Live. Our experiments show that Concerto outperforms state-of-the-art solutions, improving video quality while reducing stalling time by multi-folds under various practical scenarios.

[*]The work was done before Xiufeng Xie joined HP Labs.

MobiCom '19, October 21–25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6169-9/19/10…$15.00
https://doi.org/10.1145/3300061.3345430

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Mobile networks**.

KEYWORDS

Video Telephony; Bitrate Adaptation; Imitation Learning

ACM Reference Format:

Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, Xiaojiang Chen. 2019. Learning to Coordinate Video Codec with Transport Protocol for Mobile Video Telephony. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom'19), October 21-25, 2019, Los Cabos, Mexico.* ACM, NY, NY, USA, 16 pages. https://doi.org/10.1145/3300061.3345430

1 INTRODUCTION

Real-time interactive video communication is now an indispensable ingredient of human digital life. On the Facebook Messenger alone, 17 billion video calls happened during the year 2017 [3]. Meanwhile, emerging mobile/IoT applications, including crowd-sourced live broadcasting [42, 46, 51], cloud video-gaming [25, 45, 47, 52], tele-operation of robots or vehicles [10, 63] and real-time immersive experience sharing [2, 41, 58], continue to push the growth of video telephony traffic. On the other hand, the video quality of experience (QoE) remains unsatisfactory. Annoying blurry images, skipping frames or even video stalling often occur, especially over mobile and wireless networks [17, 64].

To improve the video telephony QoE, extensive research effort has been devoted along two separate dimensions. On one hand, modern transport layer protocols [16, 18, 19, 56, 57, 61, 65] strive to accurately estimate the time-varying end-to-end capacity of network paths. Meanwhile, video codecs keep evolving, from H.264 [5], VP8 [55] to H.265 [48] and VP9[24], to provide more bitrate options with different picture quality and frame rate, so as to accommodate instantaneous network capacity changes.

Despite the rapid evolution, we find that the video codecs and transport protocols remain highly uncoordinated in to-

day's video telephony systems, which represents one major reason for the low QoE. In particular, transport protocols usually update network capacity estimation at millisecond-level granularity (equivalent to a few RTTs), while codecs can only change video bitrate at a coarser second-level interval. The codec typically generates compressed video frames with an average bitrate that approximates the latest bandwidth estimated by the transport layer, for each forthcoming time window (denoted as Δ). However, such a straightforward interaction often fails due to the following two reasons:

(i) The objectives of the two layers are fundamentally different. While a transport protocol should be as responsive as possible to network dynamics, it is harmful for a codec to directly follow the bandwidth variation. *Firstly*, the network capacity may change within Δ, so the codec's bitrate decision, made at the beginning of Δ, may be easily outdated, leading to capacity underutilization or more severe overshoot issues, and causing large frame delay or even video stalling. *Secondly*, frequent video bitrate fluctuation breaks video smoothness, which incurs discomfort to human vision and hence degrades QoE [25, 38]. Remarkably, these two aspects entail conflicting objectives for video adaptation. A recent system, Salsify [21], shortens Δ using a fine-grained per-frame bitrate adaptation to alleviate the underuse/overshoot problem, but it may jeopardize the smoothness of video quality. In addition, to realize frame-level adaptation, Salsify has to save/restore sophisticated internal encoding states, which is not supported in existing hardware codecs [21].

(ii) The bursty application-layer traffic pattern of video telephony (*i.e.*, intermittent frame-by-frame delivery) often misleads the transport-layer's network capacity estimation, and creates a vicious cycle that leads to low network utilization. Essentially, a transport protocol infers the current network capacity by probing the path using on-going video traffic. Suppose the video codec reduces its bitrate following a temporary transport-layer congestion/loss, it will generate less (*i.e.*, more intermittent) video traffic, and thus slowing down the probing. In consequence, it usually takes excessive time (*e.g.*, 10× of the congestion duration shown in Sec. 2.3) for the video telephony application to recover from a congestion, trapping the user at low QoE unnecessarily.

We discover and validate the above incoordination problem through an unprecedented large-scale measurement campaign: *(i)* we measure over 1 million video telephony sessions of a large-scale operational video telephony system, Alibaba Taobao-Live [12]. The sessions are distributed over 749 cities worldwide and cover hundreds of ISPs and various kinds of wireless access networks. Therefore, the measurement creates a panoramic view of video telephony performance well beyond testbed/simulation experiments in prior studies [21, 64]. *(ii)* Our measurements directly log relevant performance metrics inside Taobao-Live, instead of sniffing/crawling video telephony traffic as an outsider [46, 51]. In consequence, detailed performance metrics across transport and codec layer are recorded at a *fine-grained 1-second interval* and form a dataset with a huge volume of 7.7TB. The fine-grained dataset thus captures transient path condition variations, which are of critical importance for understanding delay-sensitive video telephony performance. Quantitative analysis of the dataset reveals that the video telephony performance is far from satisfactory: over 20% sessions have low QoE across the 1 million calls. Moreover, our analysis demonstrates that the incoordination problem is pervasive, *i.e.*, widespread both temporally and spatially. For instance, it may cause frequent video stalling lasting for a few minutes on a path that could have supported a smooth 360P video telephony (Sec. 2.3).

We find that the task of coordinating video codec with transport protocol is very challenging due to the elusive network dynamics. Recent work has shown that fixed heuristic adaptation rules are insufficient even for simple file transfer over the transport layer alone [18, 56]. With video telephony, the bursty video traffic patterns and the stringent low-latency requirement further aggregate the design complexity.

In this work, we propose Concerto, a machine learning based video bitrate adaptation method to maximize video telephony QoE. We resolve the video bitrate adaptation problem through a deep imitation learning (IL) model. In the training phase, Concerto learns to map the current state (*i.e.*, historical packet loss/delay at transport layer, and also sending/receiving bitrates at the codec layer) to a target bitrate, by imitating the behavior of an expert (*i.e.*, an oracle that knows the ground-truth bandwidth and the optimal bitrate). Then in the run-time phase, the trained Concerto can independently select proper bitrates. More specifically, instead of blindly following the transport layer's path capacity estimation, Concerto reviews historical records at both the codec and transport layers, extracts high-level features of network dynamics, based on which it determines the optimal bitrates for future video content. The bitrate acts as the synergistic target for both codec and transport layers, so as to eliminate the incoordination problem. For instance, to ensure video quality smoothness or avoid overshoot, Concerto will intentionally become conservative, when it deems the path to be highly dynamic. It may refrain from increasing the video bitrate even when given a high instantaneous capacity estimation. On the other hand, Concerto will be more aggressive when detecting the aforementioned vicious cycle, and escalates the bitrate to break it.

We emphasize that Concerto's operations are fully automated by learning from experience, without using any pre-defined control rules or explicit assumptions of the network conditions. To attain such abilities, *(i)* we train the IL model using the aforementioned massive data from fine-

grained real-world video telephony traces. Such traces, with huge spatial and temporal coverage, ensure that the IL model can explore diverse environments and thus enrich its experiences. To accelerate the IL training, we develop a trace-driven packet-level simulator, which enables Concerto to "experience" every 24 hours of video telephony sessions in only 3.6 minutes. *(ii)* We custom design the IL model by incorporating domain knowledge of video telephony. Firstly, we adapt the conventional symmetric cross-entropy loss function commonly used in IL model to penalize more on bandwidth overshoot rather than underuse, since overshoot causes more damage to video QoE. Secondly, we formalize the video smoothness demand as a regularizer function, and incorporate it in the IL training objective so as to smoothen video bitrate transition.

We have implemented and incorporated Concerto into Taobao-Live, and distributed the revised Taobao-Live to real-world users for evaluation. The evaluation results demonstrate that Concerto improves video telephony QoE remarkably, reducing video stalling time by 3×, while keeping comparable quality. To our knowledge, Concerto is the first machine learning based video telephony solution with actual deployment in a large operational system. In addition, we implemented the state-of-art video telephony optimization methods Salsify [21] and another AI-driven solution [38], and validate the advantage of Concerto's customized design through extensive network traces.

Contribution: This work makes three key contributions: *(i)* Analyzing mobile video telephony performance at scale and in-depth; uncovering and quantifying the root causes for poor performance (Sec. 2). *(ii)* Designing Concerto, a customized imitation learning algorithm that can automatically generate adaptive video bitrate decisions (Sec. 3). *(iii)* Implementing, deploying and evaluating Concerto as part of a large scale operational video telephony service system (Sec. 4 and Sec. 5).

2 MOBILE VIDEO TELEPHONY PERFORMANCE IN THE WILD

In this section, we present an unprecedented panoramic analysis of mobile video telephony performance, by measuring over 1 million sessions from a major mobile video telephony service provider with worldwide customers. We investigate the root causes of QoE related issues, which motivates the design of Concerto.

2.1 Measurement Setup

Background. Existing work measured video telephony performance using in-lab testbeds or simulators [21, 56, 64] or by sniffing/crawling video telephony traffic as an outsider [46, 51, 64], which falls short of scale and temporal/spatial diversity. In this work, we directly log fine-grained end-to-end performance metrics inside an operational mobile video telephony system, without inducing extra probing traffic.

Our measurements are performed on Taobao-Live, a module inside the Taobao mobile app, which ranks as the 4_{th} most popular app in China with 600 million users [4, 7]. The Taobao-Live provides live video services worldwide. The architecture of Taobao-Live is depicted in Fig. 1. Once a video broadcaster (caller) starts a video session, her live video will be 'pushed' to a multimedia control unit (MCU) in the cloud (usually the nearest one to the caller). The MCU distributes the video in two different ways. *(i)* For callees (viewers) who request interaction with the caller, the MCU acts as a relay for the real-time video session. *(ii)* Meanwhile, the MCU periodically packs multiple frames into a chunk (with a length of $\backsim$5 seconds), and sends the chunk to a content distribution network (CDN). The CDN may redistribute the video chunk to other viewers through non-real-time applications such as video-on-demand (VoD).

In this work, we focus on the former, *i.e.*, real-time sessions. Taobao-Live builds on the web real-time communication (WebRTC) framework [8], the de facto standard for real-time Internet multimedia communications. WebRTC serves as the back-end of the mainstream browser-to-browser video telephony services (*e.g.*, Google Hangout, Slack, Facebook Messenger). It encapsulates both the transport layer and video conversation protocols. At the transport layer, it uses Google congestion control (GCC), which is based on RTP and designed for delay-sensitive traffic [16]. Different from packet-loss based congestion control protocols, GCC estimates path bandwidth mainly by examining delay variations, so as to infer a congestion in its early stage. In this way, GCC is expected to prevent queue buildup and achieve low latency for real-time traffic. At the application layer, WebRTC provides multiple codec options and Taobao-Live adopts the H.264 adaptive encoder [5]. The codec is fed with the latest available-bandwidth-estimation from GCC periodically, and generates compressed video frames with an average bitrate that approximates GCC's bandwidth estimation for the next ≈ 1 second duration.

Methodology. Inside Taobao-Live, we set multiple logging points along with the GCC transport and H.264 encoder stack, on both the client and MCU side, as illustrated in Fig. 1. Without loss of generality, we focus on video sessions happening on the caller-to-MCU path, which often involves a highly dynamic wireless first-hop, and is more likely to have lower bandwidth than the MCU-to-caller path since the wireless access uplink tends to have lower bandwidth than downlink. During each session, the client caller logs codec bitrate, GCC bandwidth estimation and other metrics (details in next subsection) averaged over each 3-second interval. It then reports the logged data in a batch to its MCU after

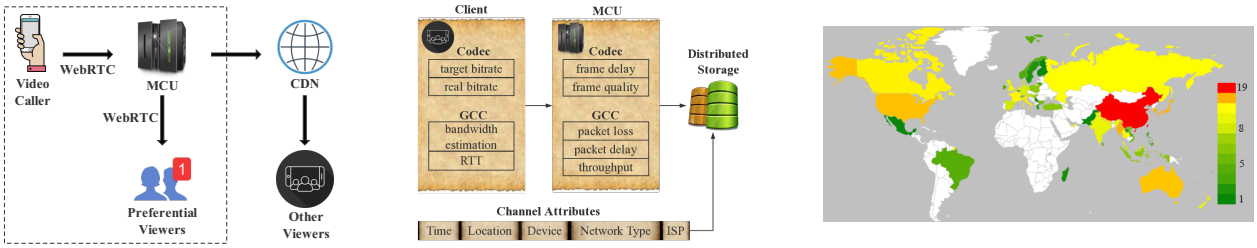


Figure 1: Left: the architecture of Taobao-Live, where live video content is captured on a caller's smartphone and 'pushed' to its associated MCU, and then is distributed to viewers. Center: the logged data inside Taobao-Live on both client and MCU sides. All logged data are stored on a distributed database. Right: we measure over 1 million sessions across 749 worldwide cities. The session amount per zone represented by the colorbar is rescaled by $\log_2$.

Table 1: Dataset statistics.

Feature	Description	Value
Sessions	sum over the dataset	1,016,652
ISP	Internet service provider	512
Country	session src country	57
City	session src city	749
Phone models	iPhone, Android	934
Wireless type	WiFi,4G,3G,LTE,2G	5
telephony hours	Sum over all sessions	1,348,548
Dataset volume	Sum over all sessions	7.7TB

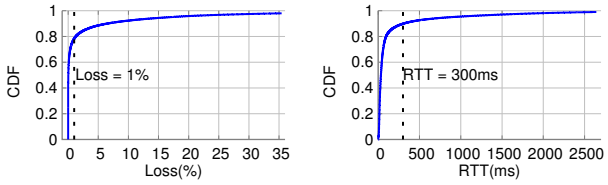


Figure 2: CDF of per-session loss and RTT.

the session. Meanwhile, the MCU, as a video receiver, logs throughput, packet loss and delay, at a more fine-grained 1-second interval.

Dataset description. Following the above methodology, we collect a measurement dataset consisting of all 1.02 million Taobao-Live sessions from Jun. 1 to June.30, 2018. The sessions are distributed worldwide as depicted in Fig. 1.

As far as we know, this is the first measurement study of mobile video telephony at such a large scale and fine granularity. Table 1 summarizes the statistics of our dataset. We note that: *(i)* The sessions are widely distributed over 57 countries (749 cites in total), 5 types of wireless access networks (WiFi, 4G, 3G, *etc.*), 512 ISPs and 934 smartphone models, with 1,348,548 hours of video calls in total. This creates sufficient real-world Internet path diversity well beyond prior testbed/simulation studies [18, 19, 21, 46, 51, 56–58, 61, 62, 64, 66]. *(ii)* The detailed measurement logs, with second-level granularity and a total volume of 7.7 TB, faithfully reflect network variation at fine timescales, including transient latency spikes and loss bursts, which are of critical importance for understanding delay-sensitive video telephony performance. For privacy preservation, our measurements do not record any video telephony content. The user IDs/IPs are all anonymized with random numbers before using the races.

2.2 Mobile Video Telephony Quality

According to the ITU G.114 standard [6], to achieve a reasonable level of QoE for video telephony, the end-to-end network path needs to maintain $\lesssim 1\%$ packet loss and $\lesssim 300$ms of round trip time (RTT). We thus evaluate the video quality through these measurable performance metrics. Fig. 2 plots the CDF of the average packet loss rate and RTT among the $\backsim$1 million sessions. We observe that, for more than 20% sessions, the average packet loss is >1%, and more than 10% sessions' average RTT exceeds 300ms.

We further analyze the metrics over space and time, and across different wireless access technologies. To simplify the exposition, we use a *poor session ratio* (PSR) to quantify the fraction of sessions with low performance in terms of average RTT or packet loss rate, *i.e.*,

$$\text{PSR} = \frac{\#sessions\ with\ RTT > 300ms\ or\ packet\ loss > 1\%}{\#all\ sessions} \quad (1)$$

Spatial patterns. In Fig. 3, we present the PSR distribution over major geographical regions that own ≥ 2000 sessions. 80.6% of the regions are domestic and 19.4% international. We can see that poor sessions are spread out spatially, instead of limited to certain regions. Specifically, all regions have a PSR $\geq 12\%$, and over half have a PSR $\geq 20\%$, *i.e.*, *one out of every five sessions experiences low QoE.*

Temporal patterns. We group all sessions into 1-hour time windows (for a session spanning multiple $\lesssim 1$ hours, we segment it into multiple windows). The timestamps follow the local time in each region. Fig. 4 plots the PSR in each window, which shows that *(i)* Low PSR happens even at the non-busy hours (1 am to 8 am), with an average of 17.5%, while that out of the duration is 28.5%. *(ii)* However, *the PSR reduction does not match traffic load reduction.* While the traffic load (*i.e.*, the number of sessions) drops by about 10×, the PSR only reduces by about 10%. The results indicate that the bitrate adaptation mechanisms are unable to harness the overall good network condition, but are heavily impacted

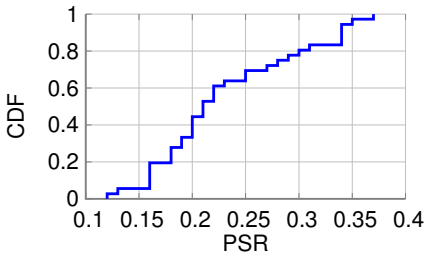


Figure 3: PSR CDF over regions.

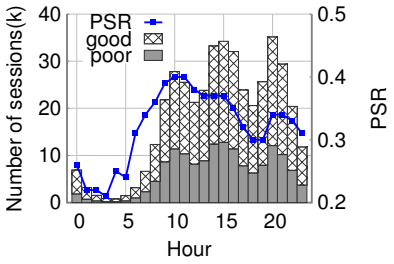


Figure 4: PSR vs. time.

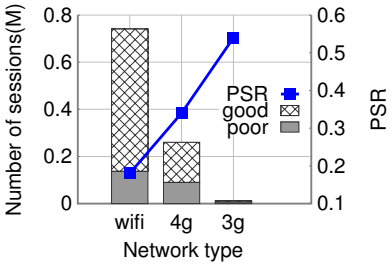


Figure 5: PSR vs. wireless access.

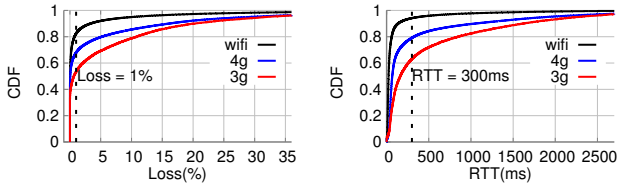


Figure 6: CDF of loss/RTT under various access.

by transient congestions, most likely inflicted by the video application itself (more in-depth analysis in Sec. 2.3).

Impact of wireless access types. Taobao-Live users access the Internet over 3 major wireless mediums: Wi-Fi, 4G and 3G (we omit the other access technologies due to very small quantity). We observe from Fig. 5 that: *(i)* Sessions over WiFi have a PSR of 18%, much better than these over 4G (34%) and 3G (54%), and occupy 73% of the sessions. *(ii)* Recent studies show that bufferbloat commonly exists in 3G/4G cellular networks [28], where overly-large buffers are deployed in the base-stations and middle-boxes, which absorbs packet loss and mitigates the bandwidth variation, but causes high latency. However, our measurements result in Fig. 6 shows that, although packet loss occurs frequently, RTT does not grow excessively, because expired frames are discarded directly by real-time telephony receiver.

Concluding remarks. Overall, our measurement demonstrates that mobile video telephony performance is far from satisfactory. At least 20% sessions have poor performance even under the most advantageous situations, such as at midnight with light traffic, or over relatively high-bandwidth access links like WiFi. The observation motivates us to investigate the root causes for the poor performance.

2.3 Understanding the Codec-Transport Incoordination Problem

We find that the root cause of poor performance lies in the lack of coordination between the transport and codec layer in Taobao-Live, and generally in all video telephony systems. Essentially, packet loss and large delay occur when an application overestimates the instantaneous bandwidth and injects excessive traffic into the network path. The amount of Taobao-Live's outgoing traffic depends on two components: GCC at the transport layer and video codec at the application layer. We find that *GCC is able to keep low packet delay and nearly-zero packet loss when running as a standalone transport protocol. However, the performance significantly deteriorates when it tries to deliver real-time video traffic.* We now showcase such problems through measurements, and analyze the underlying reasons.

GCC-alone vs. video telephony Performance. We first run GCC alone for bulk file transfer, to observe the transport-layer performance in isolation. We have implemented a packet-level trace-driven simulator with a full-fledged GCC, video bitrate adaptation framework, and also a network condition controller (implementation details in Sec. 4). The simulator can reproduce real-world network dynamics by replaying the Taobao-Live network measurement traces. It is noteworthy that: *(i)* We use simulation instead of real-world experiments mainly for efficiency: simulations speed up the evaluation by more than 400 ×, which enables us to examine performance with the massive corpus of network traces. *(ii)* The simulator faithfully models the behavior of real-world video telephony. We have compared the throughput time-series generated by our simulator and testbed experiment, respectively, under the same network conditions. We find that they have perfectly matched trend, despite slightly lower testbed throughput due to the RTP/UDP/IP-header overhead.

We first run GCC-alone and video telephony over 1000 randomly selected sessions, with a total length of 763 hours. In each session, we set the measured throughput time-series in the traces as the variable bandwidth in the simulator. We compute the average throughput, packet loss and delay of each session, and plot the CDF across all sessions in Fig. 7. Our main observations are: *(i)* The throughput of video telephony is remarkably lower than that of GCC alone (21.6% lower on average), which indicates low video quality. *(ii)* Meanwhile, the loss and delay of video telephony are much larger (*i.e.*, 4.8% vs. 0.44% in terms of loss, and 304ms vs. 141ms in terms of delay), which indicates blurred images and video stalling.

The two observations seem to be contradictory at first glance. Generally, lower throughput means lighter traffic load on the network path, and hence less packet loss and delay. However, the unique video telephony traffic pattern plus its intrinsic interaction with the transport layer causes

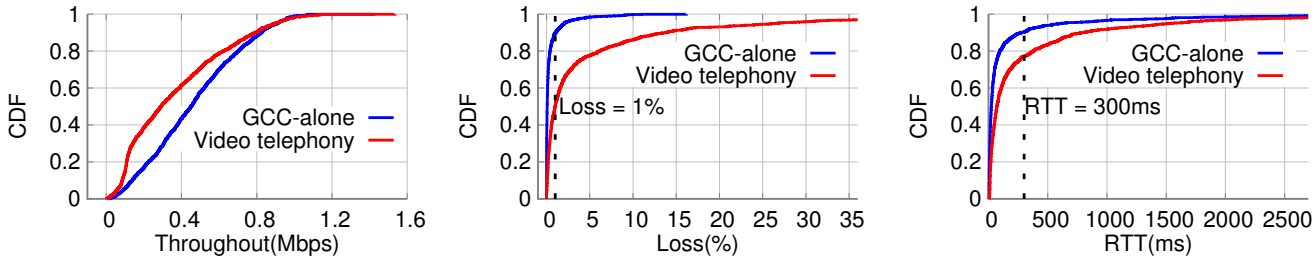


Figure 7: Performance comparison between GCC-alone and video telephony. Video traffic decreases throughput but increase loss and delay significantly.

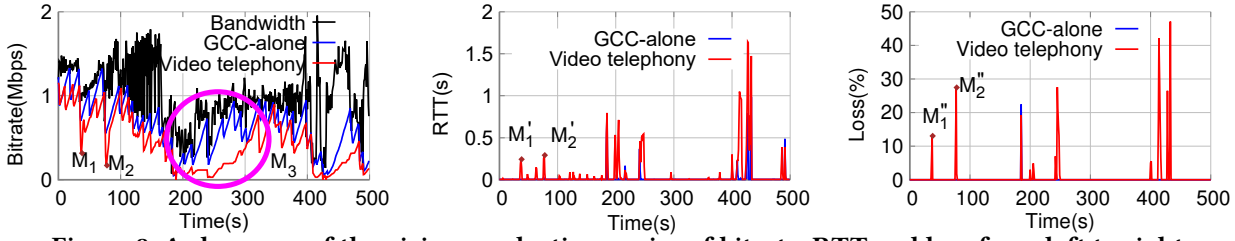


Figure 8: A showcase of the vicious cycle: time series of bitrate, RTT and loss from left to right.

the unusual detrimental effects. We showcase the issue in Fig. 8, where we plot each bitrate control decision (at millisecond timescales) during an example session. The figure reveals two microscopic effects. *(i) "Large reduction"*: Though video telephony reduces its bitrate synchronously with GCC-alone, the extent of reduction is much larger. As marked at example points of M_1 and M_2, whereas GCC decreases its bitrate estimation by 44.9% and 25.8%, video telephony backs off its bitrate by 83.6% and 58.5%, respectively. The more severe reduction stems from the fact that video telephony incurs more severe congestion just before the reduction event. In particular, the packet delay and loss during the congestion are much higher, as marked as $M_1^{'}$ and $M_2^{'}$ in the RTT plot, and $M_1^{''}$ and $M_2^{''}$ in the loss plot. *(ii) "Slow recovery"*: It takes much longer time for video telephony to recover its bitrate after congestion. For instance, video telephony spends more than 150 seconds to catch up with the current bandwidth (20 seconds for GCC-alone), during which it runs at very low bitrates, as marked in M_3. In other words, a temporary bandwidth reduction of only a few seconds may cause low video quality up to a few minutes. We find that "large reduction" and "slow recovery" pervasively exist beyond this showcase. In particular, we calculate and present the distributions of the two metrics across all sessions in Fig. 9 and Fig. 10. *Quantitatively, video telephony's bitrate reduction is* 1.2× *more severe, and its recovery time is* 4.8× *longer, compared with GCC-alone.*

We identify three underlying reasons that account for the poor performance of video telephony:

(1) The video codec cannot generate perfect bitrate to match GCC's estimation. In particular, modern codecs in video telephony use adaptive encoding techniques, which compress raw video images captured from a camera with different compression levels, and generate video streams according to a predefined target (*e.g.*, the instantaneous bandwidth estimation from the transport layer). However, we find that although the long-term average bitrate can match the target, substantial variation exists at shorter timescales. For instance, we set a target of 1 Mbps, and use an ffmpeg codec [9] to encode two 60-second-long video telephony frames. Fig. 11 plots the encoded bitrate at 1-second granularity, from which we see that: *(i)* The bitrate varies drastically, and can be as high as 2× over the average value, which will cause frequent transient congestions in video telephony. We further calculate the std. under different target bitrates ranging from 0.1 Mbps to 2 Mbps in steps of 0.1 Mbps. As shown in Fig. 12, the variation consistently occurs, and the ratio of $\frac{std}{target}$ is larger particularity for low bitrate targets. *(ii)* The variation inherently depends on video content. In Fig. 11, we observe that a video (V_2) with more static scenes has a relatively stable bitrate. Such dependency hints that a video bitrate adaptation should take into account not only the network condition, but also the codec's encoding bitrate variation.

(2) The codec's adaptation lags behind GCC. Different from bulk file transfer, mainstream video encoder cannot change its application bitrate at a short timescale. It needs at least 1 second to approximate the target bitrate[1]. By then the network bandwidth may have already changed, especially under mobility. As shown in an example trace segment in Fig. 13 the video bitrate remains high even when the ground-truth bandwidth drops at 0.5 second, and the transport layer already detects congestion. As a result, the overloaded traffic

[1]Salsify [21] is able to perform fine-grained per-frame bitrate adaptation but needs a custom-built codec.

exacerbates network congestion, causing many packet losses and large delays.

(3) Intermittent video traffic pattern interferes GCC's bandwidth estimation. The video telephony traffic is generated frame by frame. In between frames, the network path mostly remains idle, whereas a large burst of packets occurs whenever a new frame is created. On the other hand, the GCC protocol at the sender updates bandwidth estimation once an ACK (usually containing information of received packets during a short past period, *e.g.*, 5ms in WebRTC) from the receiver arrives. For GCC, the less frequently the ACK comes, the less frequently the bandwidth estimation is updated, and the slower the transport layer bitrate increases.

Quantitatively, we calculate the number of GCC updates during each bitrate recovery phases, when video telephony and GCC-alone run for all the 1000 random sessions, respectively. We plot the CDF of the counts in Fig. 14, and find that the average update counts of video telephony is only 50.6% compared with that of GCC-alone. The result validates that the intermittent video traffic pattern hinders bitrate recovery. To sum up, the update count and throughput forms a vicious cycle: the intermittent traffic pattern first causes less frequent updates and the consequent lower throughput, which further decreases update count.

2.4 Summary of Key Observations

We summarize the key measurement insights as follows: *(i)* The video telephony performance is far from satisfactory, and over 20% of $\backsim$1 million sessions experience poor performance. *(ii)* The low-quality sessions are widespread both spatially and temporally, so the poor performance is attributed to intrinsic limitations of the video telephony system itself. *(iii)* Given a certain network condition, the intermittent video telephony leads to more congestion and delay than bulk file transfer. The reason lies in the poor interaction between the codec and transport inside video telephony.

The observations motivate us to find a design that can better coordinate codec and transport layer so as to improve video telephony performance.

3 DESIGN

3.1 System Architecture

Concerto is a machine learning mechanism to enhance codec-transport interaction. As shown in Fig. 15, Concerto lies in between the codec and transport layer. Its inputs include historical transport layer information (*e.g.*, packet loss ratios, packet delay intervals), as well as codec layer information S_c (codec's bitrates at the sender side, and the throughput feedbacks from the receiver). It outputs a prediction of optimal bitrate B for the next time interval Δ (one second in our design). B is the synergistic target for both layers, *i.e.*, the codec generates video frames to approximate B and the transport layer sends out packets at the rate of B, across the Δ interval.

From a high level, Concerto is aware of both layers' status and will potentially derive unique features of each session, *e.g.*, variance of codec outputs which depends on real-time video scene complexity, intensity and endurance of path congestion which depends on network dynamics. Thus, if trained properly, it can automatically determine an optimal bitrate that matches both layers' features. For instance, Concerto can be deliberately conservative, *i.e.*, tuning down the target video bitrate B below the instantaneous transport layer capacity, if it realizes that the video content is quite dynamic, *i.e.*, the actually generated bitrate may deviate significantly from the pre-defined target (example shown in Fig. 11). On the other hand, Concerto can be more aggressive and increase B to recover from congestion quickly, if it deems the congestion as transient.

To realize the potential, we need to resolve two critical problems: *(i) How to extract the features of each session hidden in the huge status space?* and *(ii) How to determine the optimal bitrate for any feature?* A straightforward approach is to empirically set a number of thresholds to classify session status, and then map each status to a certain bitrate. However, such handcrafted design paradigm is ineffective even for the less-complicated transport-layer congestion control problem [18, 19, 27, 38, 56]. Modern Internet has evolved to be highly heterogeneous, consisting of many types of links, such as cellular/WiFi wireless links, cross-continent fiber links, intra-cloud datacenter links, all with different capacities, network latency, buffer levels, *etc.* So it is almost infeasible for a uniform set of rules to fit all these situations.

In Concerto, we exploit an imitation learning (IL) approach to address the challenges. Specifically, we design a deep neural IL model, and train it with the massive traces collected from Taobao-Live. During the training phase, the IL learns the features from various kinds of sessions and automatically generates appropriate bitrate from iterative supervised learning, guided by mimicking an expert who knows the ground-truth network condition. The IL-based Concerto is inherently flexible and generalizes well, in the sense that it learns directly from massive experiences and adjusts its actions automatically at runtime based on its observations and judgments of the scenario.

It is noteworthy that other reinforcement learning (RL) methods used in previous studies, including Q-learning and actor-critic [23, 39, 50], no longer work for video telephony. Although they have been successfully applied to VoD [38], we find that video telephony is fundamentally different. *Firstly,* video telephony has a much tighter latency constraint, whereas VoD can tolerate a multi-second buffer to amortize

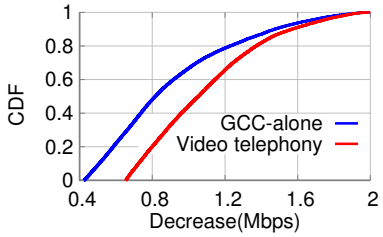


Figure 9: Comparison of bitrate decrease.

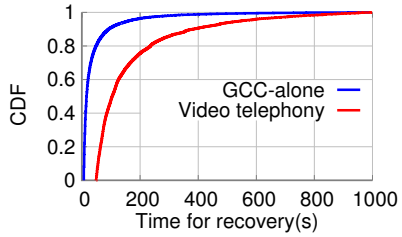


Figure 10: Comparison of recovery time.

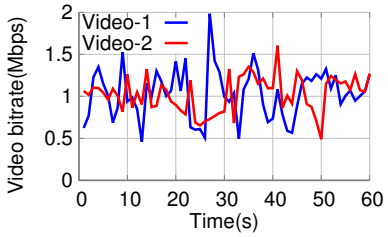


Figure 11: A showcase of video bitrate variation.

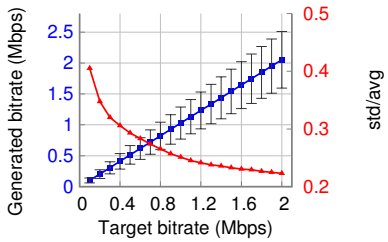


Figure 12: Video bitrate variation statistics.

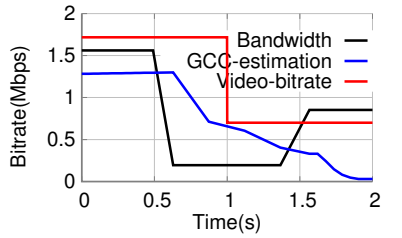


Figure 13: A showcase of the lagging effect.

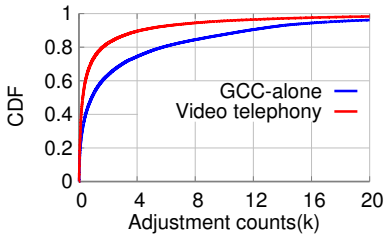


Figure 14: Comparison of update counts.

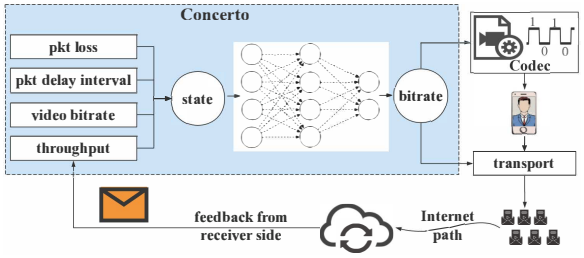


Figure 15: Concerto architecture.

short-term network/codec dynamics. Conventional RL seeks to maximize long-term cumulative rewards by planning for the far future (similar to classical dynamic programming), but video telephony's performance depends more on real-time network condition and instantaneous action. *Secondly*, RL learns indirectly from constant trial-and-error principles, which fits tasks without expert supervision. In contrast, for video telephony, we can obtain the expert's video bitrate decision (*i.e.*, the one that follows the bandwidth but remains slightly lower). Then the IL model in the training phase can learn directly from such a supervised method, thus approaching the optimal performance.

In the following, we proceed to detail the IL model and training methodologies in Concerto.

3.2 Concerto's IL Model

IL basics: The objective of an IL task is to take sequential actions in an uncertain environment to maximize a predefined metric. Formally, an IL agent interacts with an environment described by a state space $\mathcal{S}$. At any discrete time instance $t \in 0, 1, \ldots$, the agent perceives a state $s_t \in \mathcal{S}$. It decides to take an action $a_t \in \mathcal{A}$ ($\mathcal{A}$ represents the action space), and gets a reward r_t [23]. The end goal of IL is to derive an optimal state-action mapping policy $\pi^\star(s): \mathcal{S} \rightarrow \mathcal{A}$ that optimizes a certain performance metric.

To derive the optimal policy π^*, the IL agent during its training phase needs to first observe the actions of an (often human) expert, and then learns a policy that tries to mimic the expert's behavior. In practice, multiple iterations of expert guidance are necessary to incrementally optimize the policy. In Concerto, we adopt an interactive IL model, essentially a multi-round supervised learning, with workflows shown in Fig. 16. In each i-th round, the IL agent interacts with the environment with its current (non-optimal) policy π_i, and generates a set of state→action samples, denoted with $D_i = \{(s, \pi_i(s))\}$. The expert then provides supervision by mapping each s in D_i to an optimal action $\pi^*(s)$ and getting a new set of $D_i^* = \{(s, \pi^*(s))\}$. Then D_i^* is fed into a neural network to train a classifier or regressor and predict an action given any state. The policy after this round of training is expressed by the neural network model as π^{i+1}. After n loops of such supervised learning, the resulting policy π_n is expected to converge to the optimum π^* [43, 44].

Now we model video telephony bitrate adaptation as an IL task, and describe the training methodology.

Concerto input and output: We treat Concerto as an *agent* that interacts with the *environment* consisting of the network (*i.e.*, transport layer) and the application (video codec). The interaction is discretized into successive time intervals of length Δ (default to 1 second. A smaller interval does not work for the codec, and a larger interval reduces the responsiveness to network congestion.). The input state of Concerto's neural network at the t-th time interval is denoted as $s_t = (\vec{l_t}, \vec{d_t}, \vec{u_t}, \vec{v_t})$. Here $\vec{l_t}$ represents the sequence

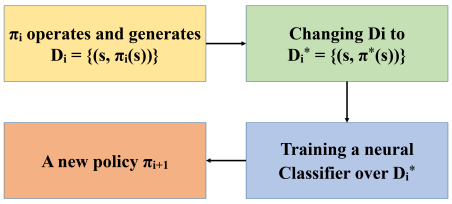


Figure 16: IL framework.

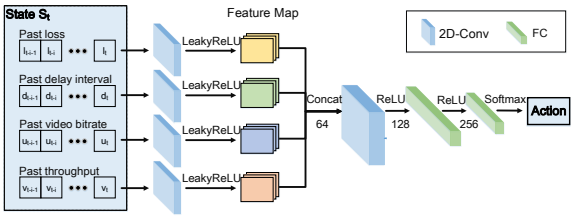


Figure 17: Concerto neural network architecture.

of packet loss statistics in the last k intervals. $\vec{d}_t$, $\vec{u}_t$ and $\vec{v}_t$ are the sequence of inter-packet delay, codec-generated sending bitrate and receiver-side throughput during the period, respectively. Except for u_t, the other three inputs are computed from periodic RTCP ACK from the receiver. Note that the ACK mechanism a standard built-in procedure in the RTP/RTCP protocol [1], widely used in today's real-time multimedia service APIs including WebRTC [8].

Concerto then maps s_t to an action a_t (*i.e.*, the target video bitrate for codec and also the sending bitrate for the transport layer) following its policy π^* learned from training, for this time interval t. After the action takes place, Concerto observes new packet loss, delay, sending bitrate and throughput statistics, and transits to a new state of $s_{t+1} = (\vec{l}_{t+1}, \vec{d}_{t+1}, \vec{u}_{t+1}, \vec{v}_{t+1})$, which in turn provokes the next action.

Concerto's neural network architecture. We represent the state-action mapping rule with a neural network (NN) parameterized by a vector θ. Fig. 17 illustrates the NN architecture which aims to represent the huge state space. Concerto first uses four independent 2D convolutional layers to distill features hidden in different input components, each layer for one of the $\vec{l}_t, \vec{d}_t, \vec{u}_t, \vec{v}_t$ inputs.

Each convolutional layer uses 3×3 kernel size with 64 filters to extract implicit features, and is followed by LeakyReLU activation function [36]. LeakyReLU can maintain non-zero gradient over the entire training phase. It is proved to effectively avoid gradient vanishing while simultaneously speeding up training. Note that the 4 input components are normalized to be the same size using zero padding, which helps to accelerate training and create the feature map with uniform size to facilitate feature combination. Next, Concerto concatenates the above four features in a late fusion manner, *i.e.*, a convolutional layer with 128 filters is adopted to learn the aggregated information fused from both the transport and application layer features. Results from the fusion layer is then aggregated in a fully-connected (FC) layer that uses 256 neurons (with a ReLU [33] activation function in a slope of 0.5). The final output is a softmax distribution function computed by the last FC layer with L2 normalization.

Neural network model choice. During the formation of our final Concerto neural model, we have also tried a simpler DNN without convolutional net (*i.e.*, fully connected net) and 1D-CNN, but their QoE (*i.e.*, average delay and picture quality as defined in Eq. (8)) is far from satisfactory. Note that the input states to our model are all 2-D. Taking the packet loss state as an example, multiple RTCP packets (default to 15) are fed back to the client within one time unit (*i.e.*, 1 second). Each RTCP packet corresponds to multiple RTP packets (default to 4), and each RTP packet brings a new packet loss ratio. So the packet loss sate is a 2-D array with dimension of 15×4. Given such inputs, it is natural to use 2-D CNN.

3.3 Training Concerto

In each i-th round of supervised learning in Concerto, the objective is to derive a new policy π_i that minimizes the expected *imitation loss* over states induced by the previous round's policy π_{i-1}, *i.e.*,

$$\pi_i = \pi_{\theta^*} = \arg\min_{\theta} \mathbb{E}_{s \sim d_{\pi_{i-1}}} [\ell(\pi_\theta(s), \pi^*(s))] \tag{2}$$

where $\ell(\pi_\theta(s), \pi^*(s))$ is the imitation loss for each state s. In Concerto, we introduce two customized designs in the training process in order to accommodate the unique characteristics of video telephony: *(i)* an imitation loss function that penalizes overshoot more than underuse and *(ii)* mechanism to smooth bitrate transition, which we detail below.

Imitation loss function. Supervised classification tasks are often trained based on cross-entropy loss $H(p, q)$:

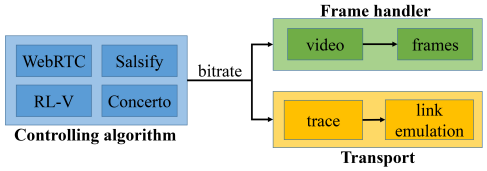
$$H(p, q) = \sum_x -p(x) \log q(x) \tag{3}$$

where q is the probability distribution of each bitrate prediction x after NN's softmax layer. x ranges from 0 to 2.5 Mbps in steps of 0.1 Mbps in Concerto. Note that $\pi_\theta(s)$ is equal to the x with the largest probability $q(x)$. p is the distribution assignment of the expert. For Concerto, large prediction deviations, even when occurring occasionally, can be detrimental to video QoE. Therefore, we propose a weighted cross-entropy to penalize such occurrences. The weights are defined as follows,

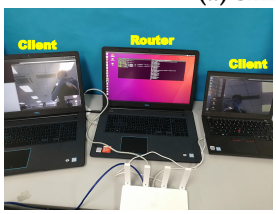
$$w(s) = \begin{cases} \|\pi_\theta(s) - \pi^*(s)\|^2 & \text{if } \pi_\theta(s) \le \pi^*(s) \\ \|\pi_\theta(s) - \pi^*(s)\|^2 + C & else \end{cases} \tag{4}$$

where the constant C reflects extra penalty for overshoot (Sec. 5 describes how we choose C). Then the imitation loss is eventually defined as:

$$\ell(\pi_\theta(s), \pi^*(s)) = w(s) \times H(p, q) \tag{5}$$



(a) Simulator.



(b) Testbed.

(c) Taobao-Live app.

Figure 18: Implementation of Concerto.

Smoothing bitrate switch. Concerto incorporates the video smoothness requirement in the training phase as a regularizer for the IL model. Suppose policy π_θ predicts the optimal bitrate in the past k intervals (up to t) as $[\pi_\theta(s(t-k)), \pi_\theta(s(t-k+1)), ..., \pi_\theta(s(t-1))]$. We mandate that the prediction at time t does not deviate much from $\phi(t,k)$, the weighted average of the historical bitrates, defined as:

$$\phi(t,k) = \sum_{i=1}^{k} 2^{-k} \times \pi_\theta(s_{t-i}) \tag{6}$$

where the closer a historical bitrate, the more important it is in $\phi(t,k)$. We incorporate $\phi(t,k)$ as a smooth regularizer in the policy-deriving objective (Eq. (2)) with a weight λ as follows,

$$\pi_{\theta^*} = \arg\min_{\theta} \mathbb{E}_{s_t \sim d_{\pi_{i-1}}} [\ell(\pi_\theta(s_t), \pi^*(s_t)) + \lambda \|\pi_\theta(s_t) - \phi(t,k)\|] \tag{7}$$

We then train the NN following the SMILE algorithm in [34]. The training algorithm iteratively updates the neural parameters θ with the stochastic gradient descent based Adam optimizer [31]), which provides provable smoothness guarantee.

Expert Label. Concerto casts imitation learning as a supervised learning model with expert guidance. In particular, an expert label is the ground-truth of instantaneous available bandwidth along the video telephony network path. The label can be easily obtained in our simulator during the model training process, without incurring any cost. The relationship between the label and video content is learned during the training phase, and then applied to run-time testing phase. We emphasize that our actual testing runs on the real video telephony system rather than in the simulator, and the actual deployment no longer needs expert labels.

Parameter choices. Many parameters impact IL performance. In Concerto, we empirically set the number of supervised-learning loops $L=4$, the proportion of expert policy in each loop $\beta=0.5$, the smooth regularizer weight $\lambda=0.5$, historical length $k=3$. We found that Concerto's performance is insensitive to most of such parameters (Sec. 5.3).

4 IMPLEMENTATION

If trained in real video telephony systems with wall-clock time, Concerto's IL model will take a formidable amount of time to converge. Instead, we train it in a full-fledged trace-driven simulator. We then incorporate the trained model into Concerto and test over both lab testbed and the operational Taobao-Live mobile app. Fig. 18 illustrates the simulator and real-world implementation, which we detail below.

The simulator consists of three modules: *(i)* A frame handler extracts frames from a real video and uses the ffmpeg encoder [9] to compress the frames, according to a target encoding bitrate. In this way, the intermittent video traffic pattern is faithfully simulated. *(ii)* In the video telephony control module, we implement Concerto, along with baseline systems including WebRTC [8], Salsify [21] and another machine learning based algorithm called RL-V (Sec. 5.1). *(iii)* The transport module decomposes each frame into RTP packets and transmits through a simulated network path whose bandwidth changes dynamically following the Taobao-Live traces we collect (Sec. 2.1). Overall, the simulator comprises over 4000 lines of Python code. In the training phase, the trace contains all types of access networks, including 3G, 4G, and WiFi, unless otherwise noted.

IL model. We implement the neural network models of Concerto and RL-V using Tensorflow framework [15] during both training and testing process. We train the NN model on a Tesla K80 [11], where each epoch takes about 15min and about 20 epochs of training are needed before convergence. Concerto's learning rate is adjusted in an exponential decay manner, initialized by 0.001 and with a slope of 0.6. We use Adam optimizer with a batch size of 32 for training.

Testbed Implementation. For small-scale controlled experiments, we built a testbed consisting of three Dell G3 Laptops running Ubuntu 16.04. Two of them run WebRTC [8] and act as a pair of video telephony nodes. We implemented the bitrate control logic of Concerto and other baseline algorithms, based on the vanilla WebRTC logic. The traffic between the telephony pair goes through the third node acting as a router, which uses Tc (traffic control, a user-space utility program used to configure the Linux kernel packet scheduler) [13] to control the network condition, so that the delay/bandwidth follows the Taobao-Live traces we collect.

Mobile APP implementation. We have also integrated Concerto into the Taobao-Live app, which is the top-4 Mobile APP in China with a user population of over 600 million [4, 7]. The imitation learning module is complied into *.pb* format and installed into mobile phones, and is called by the Taobao-Live's native C++ based bitrate control module. We implementation currently only supports iOS. The neu-

Table 2: QoE metric [21].

Name	a	b	c
QoE_p	$-6.39 * 10^{-4}$	$6.22 * 10^{-2}$	3.30
QoE_d	$-1.92 * 10^{-3}$	$1.01 * 10^{-1}$	2.67

ral network APIs used by Concerto are provided by AliNN (Alibaba's proprietary iOS deep learning framework compatible with Tensorflow). Currently, the Concerto model is 17.2 MB after compilation, using the freeze-graph model under Tensorflow [14]. The runtime computation cost of Concerto is negligible. In fact, on iPhone 7, it only takes a few milliseconds to make a bitrate decision, which is sufficient for 1-second update interval.

5 EVALUATION

5.1 Methodology

Trace-driven experiments. We use the fine-grained throughput traces of the over 1 million Taobao-Live traces (Sec. 2.1) as the varying path bandwidth in the simulator. Although the throughput value may not be exactly the same as the actual network bandwidth, it suffices to train Concerto as it captures the general trends of network dynamics. We conduct experiments over 1K sessions (about 1K hours in total) for WiFi, 4G and 3G traces separately. We use 80% of the traces for training, and the remaining 20% for test.

Baseline algorithms. We compare Concerto with three state-of-the-art video telephony frameworks:

(i) WebRTC, which is the default framework in Taobao-Live, and is also widely adopted in mainstream video telephony services, such as Google Hangouts, Slack, and Facebook Messenger [20]. Taobao-Live adopts GCC [8] to estimate the transport-layer capacity, and a H.264 codec [5] which adapts bitrate every 1 second according to the latest GCC estimation.

(ii) Salsify [21] encourages better interaction between the codec and the transport layer. It allows a codec to change its video bitrate immediately following every new estimation from the transport layer. Since standard codecs do not support such frame-level adaptation, so Salsify designs and implements a specialized codec [22].

(iii) RL-V [38, 39]. We adapt Pensieve [38], an RL based video bitrate adaptation method originally designed for VoD system, and rename it to RL-V for the new task of optimizing video telephony. The input and output of RL-V are the same as Concerto, but the neural network keeps the same as Pensive, *i.e.*, A3C [39], a conventional actor-critic based RL algorithm. Then we re-trained the A3C model using the practical video telephony traces.

QoE metric. The QoE of video telephony can be represented by a linear combination of two measurable metrics [21]: average delay (*Delay*) and picture quality (*PQ*) of video frames in the session, *i.e.*,

$$\text{QoE} = a \times Delay + b \times PQ + c \tag{8}$$

Many studies have calibrated the parameters a, b, c in the quantized QoE against subject scoring under different scenarios and preferences. We adopt the most recent version in [21], as shown in Table 2. Here QoE_p prioritizes picture quality (*e.g.*, video sessions for demonstrating commodities), and QoE_d prioritizes video delay (*e.g.*, sessions involving interactive chat or having high dynamics.)

5.2 Comparison with State-of-the-Art

We compare the performance of Concerto with baselines over three 10-hour network traces. The average normalized QoE_p and QoE_d are summarized in Fig. 19. We observe that *(i)* Concerto outperforms the three competing schemes consistently under all network conditions and QoE metrics. On average, the closest competing scheme still has a QoE gap of 23.8%, 24.9% and 34.3% under 3G, 4G and WiFi networks, respectively. *(ii)* The performance of the three baseline algorithms is quite close to each other. While RL-V is slightly better under 4G and WiFi, it becomes slightly worse under 3G. Fig. 20 and Fig. 21 further plot the CDF of each session's average throughput and frame delay. We see that Concerto achieves 54.9%, 53.9% and 37.3% throughput gain (and hence frame quality improvement) over WebRTC, Salsify and RL-V respectively, with a negligible increase on frame delay.

A microscopic showcase: We find that Concerto's gain stems from its ability to accurately estimate the instantaneous path bandwidth. In Fig. 22, we showcase a representative example of a 200-second session. We observe that Concerto follows the ground-truth bandwidth closely despite high dynamics. In contrast, WebRTC suffers from periodic and unnecessary bitrate reduction and long recovery time due to its AIMD-based adaptation mechanism. Salsify inherits the limitation because it shares the same transport layer. RL-V does not follow the bandwidth variation well, and instead exhibits large fluctuation, frequently exceeding the bandwidth. The result validates that RL designs without immediate guidance may fit for VoD or file transfer, but not agile enough for real-time video telephony.

5.3 Understanding Concerto In-depth

Impact of codec layer information. One key feature of Concerto is that it takes into account the dynamics of the codec output, in addition to network condition. To examine the significance of this design, we compare Concerto without codec bitrate input $\vec{u_t}$ and throughput input $\vec{v_t}$ (denoted with '2-state'.) against the one with full input (denoted with '4-state'). The result in Table 3 shows that the average QoE of the 2-state design is 18% lower, which validates the importance of considering the imperfect video codec, *i.e.*, it cannot generate the exact amount of video traffic at a target bitrate

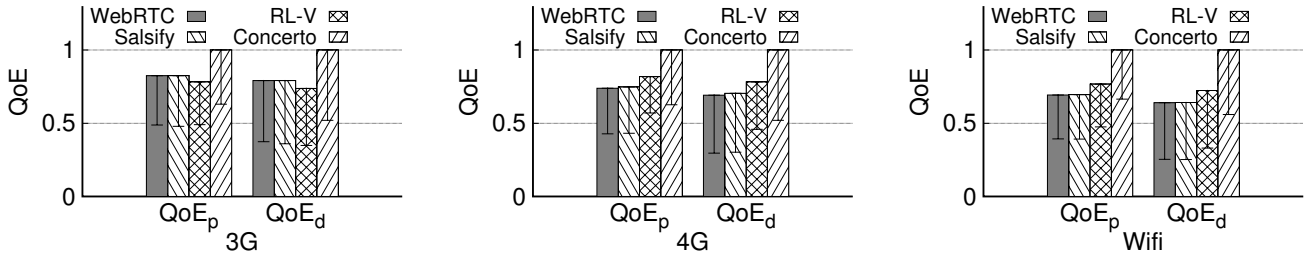


Figure 19: Concerto outperforms alternative methods over various network conditions and QoE metrics.

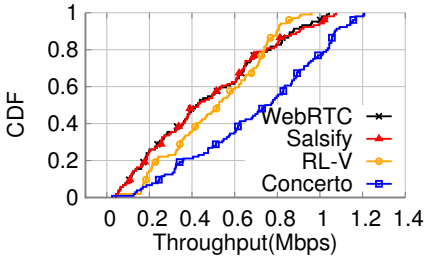


Figure 20: Throughput comparison.

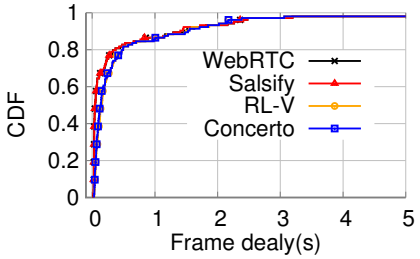


Figure 21: Frame delay comparison.

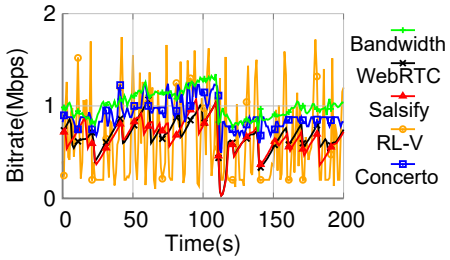


Figure 22: Bandwidth estimation.

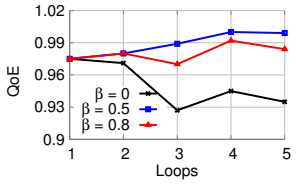


Figure 23: QoE vs. β and L.

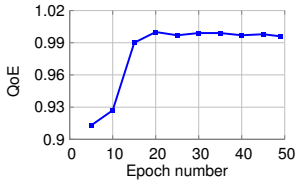


Figure 24: QoE vs. epoch.

Table 3: Impact of input and loss function.

Name	2-state	4-state	ℓ_1	ℓ_2	ℓ_3	ℓ_4
QoE_p	0.82	1	0.87	0.85	0.82	1

Table 4: Impact of trace discrepancy.

test \ train	3G	4G	WiFi
3G	**1**	0.84	0.91
4G	0.94	**1**	0.99
WiFi	0.82	0.81	**1**

in short time-scales. Meanwhile, the result also demonstrates that the neural network in Concerto can automatically learn the output patterns of the video codec and thus make better bitrate choices.

Impact of imitation loss function. We try four kinds of loss functions: ℓ_1 (the conventional cross-entropy loss, Eq. (3)), ℓ_2 (the symmetrical square loss, $C=0$ in Eq. (4)), ℓ_3 (penalizing overshoot heavily with $C=100$ in Eq. (4)), ℓ_4 (penalizing overshoot reasonably with $C=8$ in Eq. (4)). From the result normalized QoE result in Table 3, an appropriate penalization in ℓ_4 brings more than 15% gain.

Impact of the smooth regularizer. We design and incorporate a smooth regularizer in the imitation learning framework as in Eq. (7). We evaluate its impact using a smoothness metric η defined as $\eta = \frac{1}{N}\sum_{n=1}^{N}|a_{n+1}-a_n|$, where a_n denotes Concerto's predicted bitrate. We find that η of Concerto with the regularizer is 0.116 Mbps, while that without regularizer is 0.136Mbps, *i.e.*, 16.7% improvement over a randomly selected test traces totaling 30 hours. We can also observe that Concerto's bitrate is more stable than other schemes in the example Fig. 22.

Impact of trace discrepancy. We examine the generalization capabilities of Concerto, by intentionally running the training/testing in the same network and different networks, respectively. From the results in Table 4, we find that: *(i)* A model performs better (11.5% on average) when it runs in the same network with its training trace, which shows that different access networks own unique features that impact Concerto's policy training. *(ii)* Among the three access types, WiFi has the most representative behaviors, *i.e.*, the WiFi-trained model only loses 1% and 9% performance when tested on 3G and 4G, respectively.

Impact of neural network parameters. We evaluate Concerto with different neural network parameter configurations, including proportion of expert demonstration β, number of supervision loops L and training epochs. The results in Fig. 23 and Fig. 24 demonstrate that Concerto is resilient to these factors (the QoE variation $\leq$10%), compared with the impact of trace and input-state change. The parameter combination with the empirically maximum QoE (*i.e.*, $\beta=0.5$, $L=4$ and 20 epochs) is used by default in our Concerto implementation.

Table 5: Performance statistics on Taobao-Live.

Scenario	Throughput (Mbps)		Stalling time (minutes per hour)	
	WebRTC	Concerto	WebRTC	Concerto
4G static(16h)	1.34	1.04	6.1	2.1
4G walk(32h)	1.18	0.98	11.3	3.5
4G drive(4h)	1.15	1.02	19.5	2.8
WiFi static(11h)	1.24	1.03	$\backsim 0$	$\backsim 0$
WiFi walk(6h)	0.96	1.03	0.1	0.1

5.4 Real-world Deployment

We deploy Concerto on the Taobao-Live app (Fig. 18). This implementation is transparent to Alibaba's MCU servers, and can directly run on operational network infrastructure. We recruited 6 beta users to run the app in a variety of scenarios including home, office; walking inside a campus building, retail store, or outside street; and driving on an urban road. We observe that users have Wi-Fi access most of the time indoor, and 4G for most outdoor scenarios. Overall, the experiments total 300 sessions and 69 hours.

Salsify requires specialized codec [21] and RL-V's neural module is currently not supported by AliNN. So we compare Concerto against the default WebRTC in Taobao-Live. For a fair comparison, in each session, a user runs two smartphones side-by-side, with Concerto and WebRTC, respectively, so they experience identical network dynamics.

We use two performance metrics that are directly logged in Taobao-Live: the throughput representing image quality and the stalling time representing video fluency. We classify the experimental scenarios into five categories, and compare the two metrics in Table 5. We have the following observations: *(i)* Concerto has slightly lower throughput (13.1%), but it significantly reduces the stalling time (by 3×), averaging over all scenarios. *(ii)* For webRTC, the stalling time per hour increases rapidly with moving speed and up to about 20 minutes for the 4G driving test. In contrast, Concerto bounds the stalling time into 4 minutes, which greatly improves user experience. *(iii)* WebRTC performs quite well with negligible stalling in Wi-Fi scenarios, since our experiments are performed over high-bandwidth campus WiFi.

It is noteworthy that the advantage of the same Concerto module comes from different directions, when evaluated in the trace-driven experiments and app deployment. The former mainly experiences throughput (image quality) improvement and the latter gets less stalling time. We conjecture that such discrepancy attributes to the cross-traffic patterns. In practice, multiple flows compete for path capacity, while only one Concerto flow exists in the simulator and the varying bandwidth is emulated by a traffic shaper. We leave the in-depth investigation as our future exploration.

6 RELATED WORK

Video telephony measurement and optimization. Video telephony represents the most demanding application atop today's best-effort Internet service. It imposes requirements on both high throughput and low latency. Recent measurement studies consistently reveal performance limitations of video telephony, such as sluggish response to variable bandwidth and frequent video freezing [17, 62, 64, 66, 67]. Most of these measurements are performed over proprietary video chat systems like Skype. WebRTC is gaining traction recently as a full-stack open-source video telephony framework, backed by industry standardization efforts from IETF and W3C. WebRTC has enabled the design and implementation of new video telephony protocols. For instance, Salsify [21] first reports the codec-transport mismatch issue and proposes a fine-grained per-frame bit-rate adaptation for real-time video, so as to respond quickly to network condition variation. However, Salsify simply follows the transport layer's path capacity estimation and ignores the impact of the inherent traffic patterns of video telephony. In addition, Salsify requires customized codec operations such as saving/restoring internal states, which are not supported by most codecs [21]. POI360 [58] employs WebRTC to stream 360° videos, and introduces an adaptive compression and a PHY-layer-assisted bitrate control system to boost QoE. These systems are validated through controlled testbed experiments. Differently, Concerto conducts the first-of-its-kind large-scale measurement investigation of the transport-application coordination problems within a commercial video telephony system. It also represents the first AI-driven framework to resolve the problems.

Low-latency transport protocols. Driven by the demand of real-time interactive applications, extensive research had been devoted to low-latency transport protocol design. A common principle behind is to infer congestion as early as possible by examining the historical delay metrics [16, 18, 19, 57, 61, 65], or low-layer performance hints [60], in addition to packet loss metrics used in traditional congestion control protocols. While these works make significant progress on better balancing the throughput-delay tradeoff, they do not address the adverse impact of codecs on the application layer. In this work, we fill the gap by proposing Concerto to coordinate codec and transport protocol.

AI-based and data-driven protocol design. Recently, AI-based approaches have been applied to generate network protocols automatically, in contrast to the conventional handcrafted design. Examples include Remy [56], PCC [18, 19] Custard [27], QTCP [35], RL-TCP [32] for congestion control, DeepRM [37] for resource scheduling or routing, and also Pensieve [38] for VoD. Notably, Pensieve [38] focuses on bitrate selection of video chunks that are encoded a priori, and Pensieve's bitrate prediction is based on coarse-grained information (*e.g.*, average throughput for every 5-second

video chunk), without considering instantaneous RTT or Loss statistics. The input to Pensieve is the future chunk sizes, which are already known in advance for the VoD application. In contrast, Concerto performs adaptation based on much fine-grained RTT/Loss information, which is exploited to enable congestion prediction in shorter timescales. More importantly, Concerto can predict the content variation based on the historic bitrates output from the codec. To sum up, while sharing similar AI-based design principles at a high level with those works, Concerto distinguishes itself fundamentally from the following aspects: *(i)* Concerto targets real-time interactive video, which is highly sensitive to the poor interaction between codec and transport layers. Therefore, customized imitation learning architecture that can respond to path variation in real-time are designed. *(ii)* Concerto is trained and tested on a real-world video telephony application, and demonstrates the effectiveness of using realistic big data to drive large-scale network protocol and application design.

The measurement-based, data-driven design has also been adopted by video applications. For instance, CS2P [49] discovers the performance correlation among VoD flows and exploits the correlation to decide an optimal initial bitrate for a new flow. VIA [29] examines Skype VoIP worldwide performance by analyzing a dataset of 430 million calls, and revisits the classic overlay networking techniques to reroute a fraction of calls for better call quality. To our knowledge, Concerto represents the first data-driven work to optimize video telephony, which imposes stringent latency and throughput constraints.

Transport optimization for Video-on-Demand (VoD) applications. Similar transport-application mismatch problems have been studied in the VoD domain, for instance, in [26, 30, 59] and references therein. However, Concerto is different in two major aspects: *(i)* Those works focused on the mismatch between the HTTP protocol and the TCP protocol in video streaming. Whereas Concerto focuses on the mismatch between the codec and the TCP protocol in real-time video telephony. *(ii)* The design space to solve the two mismatches is quite different. In particular, VoD clients usually have a buffer lasting for dozens of seconds or even minutes, so they are much less sensitive to short-timescale network condition variations. In addition, much information of VoD (*e.g.*, the size of every future video chunks) is known in advance when doing optimization. While in video telephony, the content is generated in real-time. Concerto needs to predict content dynamics and perform responsive adaptation under very tight latency constraint.

7 DISCUSSION

IL model complexity. In Concerto, we adopt a imitation learning model, and train the model using a 2-D convolutional neural network. We believe that the complexity is necessary in order to have the modeling and prediction capacity. In our experiments, we have compared Concerto against simpler state-of-the-art solutions, such as RL-V, which uses A3C reinforcement learning, instead of imitation learning. Besides, we have also tried simpler DNN without convolutional nets (*i.e.*, fully connected net) and 1D-CNN, but their QoE is far from satisfactory. We emphasize that though training an imitation learning model costs a long time, it is very fast when applying the model for online optimization. In fact, the run-time decision-making overhead is on the order of milliseconds (Sec. 4).

Storage and energy efficiency. The current Concerto module occupies 17MB storage, which is a bit bulky for smartphones. The extra energy cost of Concerto still needs to be quantified, although we expect it will not be an issue considering its millisecond-level run-time computational load (Sec. 4). We are working on optimizing the Concerto code, and making it available to millions of Taobao-Live users.

Advanced neural models and their interaction. Concerto adopts a simple imitation learning model [44] as a starting point, but it already shows a significant gain over reinforcement learning models previously used in congestion control or VoD systems. New IL architectures keep evolving [40, 53, 54], and may further improve video telephony performance. Moreover, it is also interesting to see how Concerto interacts with conventional video telephony adaptation algorithms or other AI-driven algorithms when competing on the same Internet paths, which is left for future exploration.

8 CONCLUSION

Through a large measurement campaign over 1 million video telephony sessions, we have uncovered the codec-transport incoordination problem which accounts for the low QoE. We then designed and deployed a data-driven, imitation learning based algorithm, Concerto, which can determine the synergistic optimal bitrate for both codec and transport layers, so as to maximize the QoE. We found that Concerto can effectively and rapidly track the network bandwidth changes, and its QoE outperforms the state-of-the-art solutions over a broad set of network conditions.

ACKNOWLEDGMENTS

We appreciate the insightful feedback from the anonymous reviewers and our shepherd Prof. Sunghyun Choi who helped improve this work. This project was supported by NSFC (61720106007, 61772084, 61832010, 61532012), the 111 Project (B18008) and Alibaba AIR project. Xinyu Zhang was supported in part by US NSF funding (CNS-1350039, CNS-1506657, CNS-1518728 and CNS-1617321), and by a Google Faculty Award.

REFERENCES

[1] 2003. RTP: A Transport Protocol for Real-Time Applications. https://tools.ietf.org/html/rfc3550.

[2] 2016. Exo360 Drone Shoots VR Content on the Fly. https://newatlas.com/exodrone360-vr-content/43854/.

[3] 2017. Facebook Messenger doubles number of video chats to 17 billion in 2017. https://www.theverge.com/2017/12/13/16772704/facebook-messenger-17-billion-video-chats-2017.

[4] 2018. 30 Amazing Taobao Statistics and Facts. https://expandedramblings.com/index.php/taobao-statistics/.

[5] 2018. H.264 : Advanced video coding for generic audiovisual services. https://www.itu.int/rec/T-REC-H.264.

[6] 2018. QoS Requirements for Video. http://www.ciscopress.com/articles/article.asp?p=471096&seqNum=6.

[7] 2018. Top mobile apps in China in Q2 2018. https://www.chinainternetwatch.com/26542/mobile-apps-q2-2018/.

[8] 2018. WebRTC Home. https://webrtc.org/.

[9] 2019. ffmpeg. https://www.ffmpeg.org/.

[10] 2019. Long Distance Tele-Operation system for remote control of unmanned ground vehicles. https://newatlas.com/long-distance-tele-operation-of-ugvs/19423/.

[11] 2019. Nvidia Tesla. https://www.nvidia.cn/object/tesla-servers-cn.html.

[12] 2019. TAOBAO LIVE BROADCAST: 500 BILLION IN THE NEXT THREE YEARS. https://www.marketingtochina.com/taobao-live-broadcast-500-billion-in-the-next-three-years/.

[13] 2019. Tc (Linux). https://en.wikipedia.org/wiki/Tc_(Linux).

[14] 2019. Tensorflow source code. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools.

[15] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[16] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. 2017. Congestion Control for Web Real-Time Communication. *IEEE/ACM Transactions on Networking* 25, 5 (Oct 2017), 2629–2642. https://doi.org/10.1109/TNET.2017.2703615

[17] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. 2011. Skype Video Congestion Control. *Comput. Netw.* 55, 3 (Feb. 2011), 558–571. https://doi.org/10.1016/j.comnet.2010.09.010

[18] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 395–408. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong

[19] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *NSDI*. USENIX Association, 343–356.

[20] Sam Dutton. 2012. Getting Started with WebRTC. *HTML5 Rocks* 23 (2012).

[21] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*.

[22] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[23] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. 2018. An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning* 11, 3-4 (2018), 219–354. https://doi.org/10.1561/2200000071

[24] A. GRANGE, DE RIVAZ P., and J. HUNT. 2016. VP9 Bitstream and Decoding Process Specification version 0.6. http://www.webmproject.org/vp9/.

[25] Chun-Ying Huang, Cheng-Hsin Hsu, De-Yu Chen, and Kuan-Ta Chen. 2013. Quantifying User Satisfaction in Mobile Cloud Games. In *Proceedings of Workshop on Mobile Video Delivery (MoViD'14)*. Article 4, 6 pages.

[26] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference, IMC '12, Boston, MA, USA, November 14-16, 2012*. 225–238.

[27] Nathan Jay, Noga H. Rotman, Philip Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2018. Internet Congestion Control via Deep Reinforcement Learning. *CoRR* abs/1810.03259 (2018). arXiv:1810.03259 http://arxiv.org/abs/1810.03259

[28] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling Bufferbloat in 3G/4G Networks. In *Proc. of ACM Internet Measurement Conference (IMC)*.

[29] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A. Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, and Hui Zhang. 2016. Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 286–299. https://doi.org/10.1145/2934872.2934907

[30] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Trans. Netw.* 22, 1 (2014), 326–340.

[31] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[32] Yiming Kong, Hui Zang, and Xiaoli Ma. 2018. Improving TCP Congestion Control with Machine Intelligence. In *Proceedings of the 2018 Workshop on Network Meets AI & ML (NetAI'18)*. ACM, New York, NY, USA, 60–66. https://doi.org/10.1145/3229543.3229550

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[34] Hoang Minh Le, Andrew Kang, Yisong Yue, and Peter Carr. 2016. Smooth Imitation Learning for Online Sequence Prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 680–688. http://jmlr.org/proceedings/papers/v48/le16.html

[35] W. Li, F. Zhou, K. R. Chowdhury, and W. M. Meleis. 2018. QTCP: Adaptive Congestion Control with Reinforcement Learning. *IEEE Transactions on Network Science and Engineering* (2018), 1–1. https://doi.org/10.1109/TNSE.2018.2835758

[36] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. 3.

[37] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics*

in Networks (HotNets '16). ACM, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[38] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 197–210. https://doi.org/10.1145/3098822.3098843

[39] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 1928–1937. http://jmlr.org/proceedings/papers/v48/mniha16.html

[40] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. *arXiv preprint arXiv:1806.05635* (2018).

[41] Sergio Orts-Escolano, Christoph Rhemann, Sean Ryan Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles T. Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien P. C. Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. 2016. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*. 741–754.

[42] Haitian Pang, Zhi Wang, Chen Yan, Qinghua Ding, and Lifeng Sun. 2017. First Mile in Crowdsourced Live Streaming: A Content Harvest Network Approach. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017 (Thematic Workshops '17)*. 101–109.

[43] Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and Imitation Learning via Interactive No-Regret Learning. *CoRR* abs/1406.5979 (2014). arXiv:1406.5979 http://arxiv.org/abs/1406.5979

[44] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. 627–635. http://proceedings.mlr.press/v15/ross11a/ross11a.pdf

[45] Ryan Shea, Jiangchuan Liu, Edith C. H. Ngai, and Yong Cui. 2013. Cloud gaming: architecture and performance. *IEEE Network* 27, 4 (2013), 1–0.

[46] Matti Siekkinen, Enrico Masala, and Teemu Kämäräinen. 2016. A First Look at Quality of Mobile Live Streaming Experience: The Case of Periscope. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. 477–483.

[47] Ivan Slivar, Lea Skorin-Kapov, and Mirko Suznjevic. 2016. Cloud Gaming QoE Models for Deriving Video Encoding Adaptation Strategies. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. Article 18, 12 pages.

[48] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (Dec 2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191

[49] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 272–285. https://doi.org/10.1145/2934872.2934898

[50] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction*. MIT Press. http://www.worldcat.org/oclc/37293240

[51] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y. Zhao. 2016. Anatomy of a Personalized Livestreaming System. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*.

[52] H. Wang, T. Li, R. Shea, X. Ma, F. Wang, J. Liu, and K. Xu. 2018. Toward Cloud-Based Distributed Interactive Applications: Measurement, Modeling, and Analysis. *IEEE/ACM Transactions on Networking* 26, 1 (2018), 3–16.

[53] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 2018. Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation. *arXiv preprint arXiv:1811.10092* (2018).

[54] Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. 2017. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*. 5320–5329.

[55] P. WILKINS, Y. XU, L. QUILLIO, J. BANKOSKI, J. SALONEN, and J.. KOLESZAR. 2015. VP8 Data Format and Decoding Guide.

[56] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 123–134. https://doi.org/10.1145/2486001.2486020

[57] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*.

[58] Xiufeng Xie and Xinyu Zhang. 2017. POI360: Panoramic Mobile Video Telephony over LTE Cellular Networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*. 336–349.

[59] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *Proceedings of ACM MobiCom*.

[60] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. 2017. Accelerating Mobile Web Loading Using Cellular Link Information. In *Proceedings of ACM MobiSys*.

[61] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '13)*. 347–360.

[62] Y. Xu, C. Yu, J. Li, and Y. Liu. 2014. Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype. *IEEE/ACM Transactions on Networking* 22, 3 (June 2014), 826–839. https://doi.org/10.1109/TNET.2013.2260354

[63] A.W.W. Yew, S.K. Ong, and A.Y.C. Nee. 2017. Immersive Augmented Reality Environment for the Teleoperation of Maintenance Robots. *Procedia CIRP* 61 (2017), 305 – 310. https://doi.org/10.1016/j.procir.2016.11.183 The 24th CIRP Conference on Life Cycle Engineering.

[64] C. Yu, Y. Xu, B. Liu, and Y. Liu. 2014. "Can you SEE me now?" A measurement study of mobile video calls. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 1456–1464. https://doi.org/10.1109/INFOCOM.2014.6848080

[65] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 509–522. https://doi.org/10.1145/2785956.2787498

[66] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. 2012. Profiling Skype video calls: Rate control and video quality. In *2012 Proceedings IEEE INFOCOM*. 621–629. https://doi.org/10.1109/INFCOM.2012.6195805

[67] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang. 2013. Modeling and Analysis of Skype Video Calls: Rate Control and Video Quality. *IEEE Transactions on Multimedia* 15, 6 (Oct 2013), 1446–1457. https://doi.org/10.1109/TMM.2013.2247988