POI360: Panoramic Mobile Video Telephony over LTE Cellular Networks

Xiufeng Xie
University of Wisconsin-Madison
xiufeng@ece.wisc.edu

Xinyu Zhang
University of California San Diego
xyzhang@ucsd.edu

ABSTRACT

Panoramic or 360° video streaming has been supported by a wide range of content providers and mobile devices. Yet existing work primarily focused on streaming on-demand 360° videos stored on servers. In this paper, we examine a more challenging problem: Can we stream real-time interactive 360° videos across existing LTE cellular networks, so as to trigger new applications such as ubiquitous 360° video chat and panoramic outdoor experience sharing? To explore the feasibility and challenges underlying this vision, we design POI360, a portable interactive 360° video telephony system that jointly investigates both panoramic video compression and responsive video stream rate control. For the challenge that the legacy spatial compression algorithms for 360° video suffer from severe quality fluctuations as the user changes her region-of-interest (ROI), we design an adaptive compression scheme, which dynamically adjusts the compression strategy to stabilize the video quality within ROI under various user input and network condition. In addition, to meet the responsiveness requirement of panoramic video telephony, we leverage the diagnostic statistics on commodity phones to promptly detect cellular link congestion, hence significantly boosting the rate control responsiveness. Extensive field tests for our real-time POI360 prototype validate its effectiveness in enabling panoramic video telephony over the highly dynamic cellular networks.

CCS CONCEPTS

• **Networks** → **Cross-layer protocols**;

KEYWORDS

360 degree video; LTE; cellular network

CoNEXT '17, Incheon, Republic of Korea

DOI: 10.1145/3143361.3143381

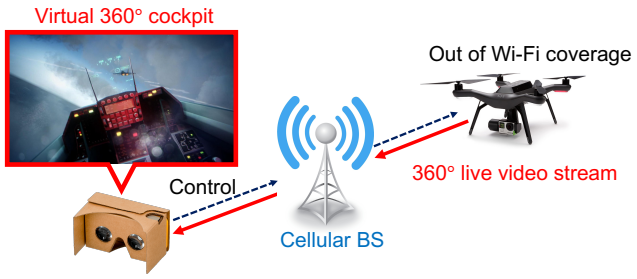


Figure 1: Example application scenario for portable interactive 360° video telephony over cellular networks.

1 INTRODUCTION

Panoramic videos, also known as 360° videos, is gaining traction in the consumer electronics and mobile application ecosystem. Many video content providers [9, 45] recently started to stream 360° videos, which are usually captured by omnidirectional cameras, and can be viewed by virtual reality head-mounted-displays (VR HMDs), such as Oculus Rift, Google Daydream, HTC VIVE, and Samsung GearVR. Besides fetching 360° videos stored on a server, the emerging mobile 360° stereoscopic cameras [22, 25, 29, 34], combined with the mobile HMDs, can potentially enable a new wave of *interactive 360° video* services that capture & stream 360° videos in real-time. Some examples include elevating current 2D video chat into 360° video chat, panoramic live broadcasting for outdoor events, and flying a drone remotely as if sitting inside a virtual cockpit (Fig. 1). However, existing works mainly investigate on-demand 360° video streaming from contend providers, while the area of interactive 360° video telephony remains mostly unexplored.

In this paper, we present POI360, the first POrtable Interactive 360° video telephony system running over existing LTE cellular network. Two major challenges emerge in realizing POI360: *(i) Portability.* To perform ubiquitous live streaming even out of the Wi-Fi coverage, POI360 relies on the mobile broadband. However, a 360° video contains views in every direction and must entail high resolution for the near-eye display of HMD, which translates into a bitrate of tens of Mbps and drastically overloads the LTE uplink (median bandwidth around 2.2 Mbps [13]). *(ii) Interactivity.* POI360 confronts an intrinsic conflict between the strict latency requirement of interactive video and the highly dynamic cellular network condition. The situation is even more challenging when the

heavy 360° video traffic pushes the cellular link to its limit, as a fluctuating link bandwidth can easily fall below the high traffic load. POI360's rate control must detect this and reduce the traffic rate promptly to prevent video freezes.

To meet these challenges, POI360 design achieves both efficient compression to curtail the overwhelming bitrate of 360° video stream, and responsive rate control to facilitate smooth real-time video telephony. *(i)* Although existing works on 360° video compression [11, 24, 30, 33] can reduce the traffic load by prioritizing the encoding quality of the region-of-interest (ROI) in a 360° video frame, the highly dynamic cellular network latency may easily fail such ROI-based compression, as the sender's ROI knowledge may lag behind the remote user's actual ROI, causing mismatches between the sender-allocated encoding quality and user-perceived display quality, and thus severe quality fluctuations (§3). POI360 answers this challenge with an adaptive spatial compression algorithm – It dynamically tunes the encoding quality distribution across the 360° panorama based on the ROI update responsiveness, thus guarantees a stable user-perceived quality even during user ROI change. *(ii)* To boost the responsiveness of rate control, since cellular uplink is the typical network bottleneck for interactive video (§3), POI360 monitors the cellular device's uplink firmware buffer occupancy to instantaneously detect any uplink congestion. Upon congestion, POI360 degrades its uploading bitrate to the windowed-sum of cellular uplink transport block size (TBS), which is the guaranteed bandwidth.

We have implemented a POI360 prototype over commercial LTE networks. Targeting VR HMD using smartphone as screen, POI360 runs in mobile browsers, and our implementation integrates WebRTC [18] which serves as the back-end of most interactive video services [40], and WebGL, which converts the 360° video stream to the HMD-compatible stereoscopic view in the browser. We implement POI360's cellular-link-informed rate control in the WebRTC module of the Chromium browser. The required cellular physical-layer statistics, including the firmware buffer level and TBS, are read from the diagnostic interface on commodity phones. Therefore, POI360 requires no hardware modification and is readily deployable. Our run-time experiments demonstrate that POI360's adaptive spatial compression algorithm significantly outperforms existing benchmark algorithms, and POI360's rate control is much more responsive than conventional solutions that rely on end-to-end delay/loss metrics. We have also conducted field tests under various locations and mobility levels, which confirm POI360's effectiveness in enabling real-time interactive 360° video services over LTE.

The contributions of POI360 are summarized as follows:

(i) To the best of our knowledge, POI360 is the first portable interactive 360° video telephony system customized for existing LTE cellular network infrastructure.

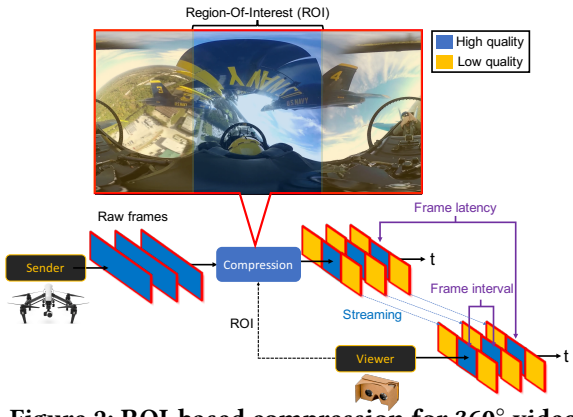


Figure 2: ROI-based compression for 360° video.

(ii) POI360 features a novel adaptive spatial compression scheme for 360° video, which dynamically tunes the compression strategy to fit the varying cellular network conditions.

(iii) POI360's awareness of cellular link statistics enables prompt adaptation to facilitate smooth 360° video telephony.

2 BACKGROUND FOR 360° VIDEO

360° videos are typically shot using an array of cameras [32] whose field-of-view (FoV) together covers 360°. The video frames captured from each camera at the same moment are stitched and then projected onto a sphere, which is further mapped to a planar format as one 360° video frame. This is known as equirectangular projection and the world map is a classical example. To further compress the video data, alternative conversion schemes instead project the sphere into a cube [8] or pyramid [10]. Nonetheless, the bitrate of a 360° video (*e.g.*, tens of Mbps for 4K video [11]) typically exceeds the bandwidth of wireless links [15].

To alleviate the bandwidth crunch, the region-of-interest (ROI) streaming has been proposed [6, 24, 33, 35, 39]. Human eyes have limited FoV, and the vision resolution is high at the central visual field (fovea) but drops almost quadratically with distance from the center. Accordingly, the resolution of regions near the visual periphery or even out of FoV can be compressed more aggressively. As shown in Fig. 2, by estimating the viewer's foveation using head-orientation sensors inside a VR HMD, the video sever can selectively compress the video quality—the ROI will be delivered with high resolution while the remaining regions will have lower quality. Such ROI streaming mechanisms can bring the bandwidth cost to a level that can be supported even in cellular networks [30]. Besides orientation sensors, alternative approaches can further narrow down the ROI, *e.g.*, using an eye-tracking camera to track the viewer's focus [35], using semantic analysis of the video frame content or crowd-sourced viewing statistics [11, 28, 30]. Although this work focuses on the head-orientation-based ROI compression, our solution can be potentially applied to such systems.

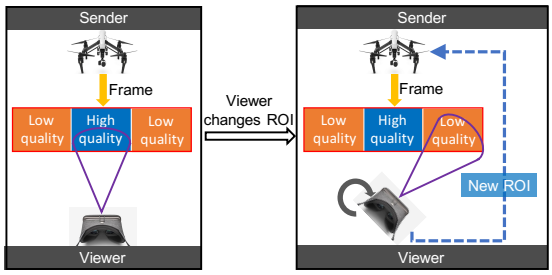


Figure 3: Problem for ROI-based compression: viewer ROI change causes user-perceived quality fluctuation.

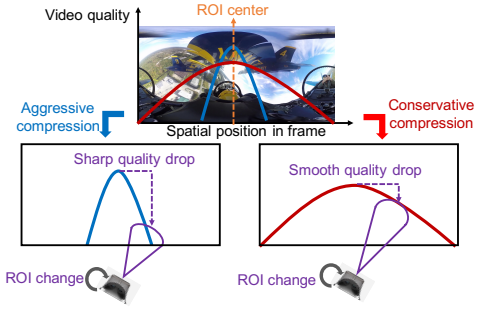


Figure 4: Trade-off between compression ratio and user-perceived quality stability during ROI update.

3 CHALLENGES & SOLUTION SPACE

In this section, we investigate the challenges for realizing portable interactive panoramic video telephony over cellular networks, and explore the corresponding solution spaces.

3.1 ROI Quality Fluctuation.

Challenge I. When running over highly dynamic cellular networks with much longer and unstabler latency than the wireline access networks [46], ROI-based spatial compression causes frequent user-perceived video quality fluctuations for the ROI region as the user view changes. As shown in Fig. 3, due to the network latency of ROI feedback, the panoramic video sender and viewer may have inconsistent knowledge of the viewer's current ROI, thus there can be a temporary mismatch between the sender-allocated quality and user-perceived quality for the ROI region whenever the user view changes, causing quality fluctuations and user sickness. In §6, we further verify this phenomenon through real-time experiments (Fig. 12b). We note that motion-based ROI prediction [2, 21] cannot fully solve this problem as its prediction horizon is still shorter than the end-to-end latency of mobile interactive video sessions (§8).

Solution space I. To mitigate the quality fluctuation, we need to address the trade-off between the compression aggressiveness and the stability of ROI quality. An aggressive compression mode can focus the quality on a small ROI while minimizing the non-ROI bitrate. But it is sensitive to the ROI changes as the user's FoV can easily enter the non-ROI region. So it applies only to networks with small feedback latency. On the other hand, a conservative compression mode that distributes the quality more evenly across spatial regions is suitable for high-latency networks in order to keep stable video quality in ROI region even under laggy ROI feedback, but has to sacrifice the quality itself to fit the entire frame within limited network bandwidth. To make the situation worse, cellular networks latency is also highly dynamic due to the varying wireless environment, causing unstable responsiveness of the end-to-end ROI update.

To resolve these issues, instead of sticking to a fixed compression mode, we can dynamically switch across different modes based on the responsiveness of ROI update, as shown in Fig. 4. In this way, we can aggressively reduce the traffic load when current network condition allows the sender to responsively update its ROI knowledge, and conservatively compress the stream to maintain a stable video quality when the end-to-end ROI update becomes sluggish.

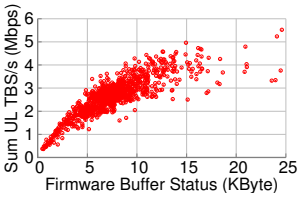


Figure 5: The relation between firmware buffer occupancy and TBS.

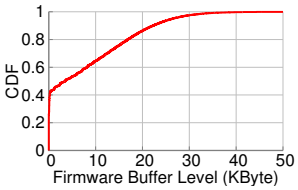


Figure 6: Uplink firmware buffer level under WebRTC rate control.

3.2 Video Rate Control Responsiveness

Challenge II. The rate control algorithms must responsively adapt the panoramic video encoding bitrate based on network conditions, to avoid congestion and video freeze. However, existing video bitrate adaptation algorithms for interactive videos commonly adopt end-to-end network condition estimation (§2).Such strategies fail to meet the responsiveness requirement of interactive panoramic video, especially over highly dynamic cellular networks (experimental evidences will be shown in §6).

Solution space II. For panoramic video telephony, cellular uplink is typically the network bottleneck, but needs to sustain the same traffic load as the downlink or wireline segments. Therefore, the sender can bypass the sluggish end-to-end metrics, and directly execute congestion detection and bandwidth estimation for the cellular uplink alone.

3.3 Uplink Bandwidth Underutilization.

Challenge III. Due to the use of proportional fair scheduling in LTE networks [44], the user device's uplink firmware buffer has an unique feature – its service rate depends on its own buffer occupancy. Fig. 5 showcases this effect, where we measure the buffer size and uplink throughput R_{phy} (indicated by the TBS) on an LTE smartphone. With small buffer

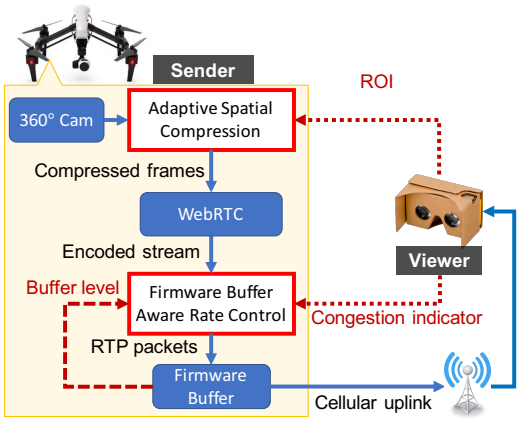


Figure 7: POI360 system architecture.

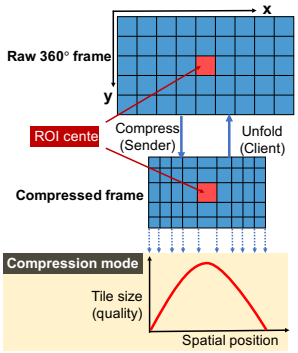


Figure 8: POI360 video compression model.

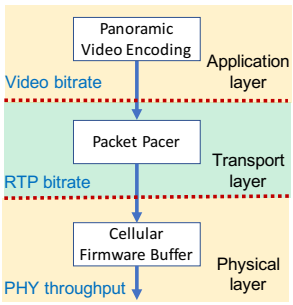


Figure 9: POI360 rate control model.

size, the R_{phy} increases almost linearly with the buffer size, and saturates when reaching the uplink bandwidth. Unaware of this effect, generic video rate control frames, like WebRTC, simply set the RTP sending rate R_{rtp} to match the video bitrate R_v. However, a temporary uplink bandwidth surge can drain the buffer quickly, and the RTP sender may leave the buffer empty as it needs to follow the R_v cap, even if there exists pending traffic at the video application layer. As an evidence, Fig. 6 plots the CDF of uplink buffer level when streaming a 4K panoramic video through an LTE phone (more details about the setup are in §5). We see that the uplink buffer becomes empty for 40% of the time, even though the traffic always exceeds the available bandwidth.

Solution space III. The knowledge of the uplink firmware buffer level opens a new opportunity to address this challenge. In particular, the sender can push the firmware buffer level to a "sweet" region which is far from congestion but still high enough to harness the bandwidth provisioned by the basestation's uplink scheduling. As the video encoding bitrate R_v should not be changed too fast to avoid quality fluctuations that may degrade user experience, we keep updating the RTP sending rate R_{rtp} based on the instantaneous buffer level to keep it in the "sweet" region.

4 POI360 SYSTEM DESIGN

In this section, we present our POI360 system design to exploit the solution space discussed in §3. As outlined in Fig. 7, the POI360 sender first applies *adaptive spatial compression* to perform ROI-based spatial compression to the input 360° video stream while adapting the compression mode based on current network condition, the ROI-compressed stream is then encoded by the legacy WebRTC framework[18] (by default the VP8 encoder) which performs temporal compression to the frames. The sender then uses *firmware buffer aware rate control* to efficiently deliver the encoded stream to the remote viewer wearing a VR HMD. Packet losses and rebuffering during streaming are handled by WebRTC's builtin mechanisms[14]. In what follows, we introduce POI360's system model (§4.1), and then describe its two major design modules (§4.2 and §4.3) in more detail.

4.1 System Model

360° frame compression. As illustrated in Fig. 8, POI360 compresses 360° video stream at the video frame level. Each raw 360° video frame is first spatially segmented into small tiles, which are then compressed separately based on their relative position *w.r.t.* the user's current ROI center. Finally, all tiles of a video frame are stitched together and sent through the cellular uplink. Based on Fig. 8, we define several compression parameters to be used across the paper. Let i denote a tile's position in the x-axis while j denote its position in the y-axis in a 360° video frame. The center of the user's current ROI is a tile whose position is denoted as $r = (i^*, j^*)$. We define *compression level* l_{ij} as the ratio of tile size before and after compression, and *compression matrix* $\boldsymbol{L}$ is the matrix of compression level l_{ij} for all tiles. We further define *compression mode* $\mathcal{F}$ as the mapping that determines the compression level l_{ij} of each tile based on the distance between its position (i, j) and the current ROI center $r = (i^*, j^*)$, *i.e.*, $l_{ij} = \mathcal{F}(i - i^*, j - j^*)$. The basic principle behind POI360's compression mechanism is to use higher compression level for the tiles further away from the ROI center r, and the center itself should always have the lowest compression level l_{min}. In our current POI360 prototype, we implement the compression by scaling down the width and length of each tile based on its distance to the ROI center, *i.e.*, the compression ratio for the tile width is C^{i-i^*} and similarly C^{j-j^*} for the tile height, where $C > 1$ is a constant value controlling the compression aggressiveness for this compression mode. As a result, the compression mode is essentially defined as:

$$l_{ij} = \mathcal{F}(i - i^*, j - j^*) = C^{(i-i^*)+(j-j^*)} \quad (1)$$

Under a certain compression mode, when the ROI center shifts with the user's head position, the compression matrix will be updated following Eq. (1), which is essentially a cyclic shift based on the shift of ROI center.

360° video stream rate control. As shown in Fig. 9, POI360 uses a cross-layer rate control model at the video sender, which contains two important buffers: the video buffer in the application layer, and the firmware buffer in the LTE PHY layer. Below are the crucial model parameters:

360° video encoding bitrate (R_v) determines the bitrate of the compressed video source, *i.e.*, traffic injection rate (video quality) into the video buffer.

RTP sending rate (R_{rtp}) in the transport layer parameter controlling the outgoing traffic of the video buffer and the input traffic into the cellular firmware buffer.

PHY-layer throughput (R_{phy}) determines the outgoing traffic of the cellular firmware buffer. R_{phy} is controlled by the BS's uplink scheduling policy. However, as will be clarified soon, POI360 can indirectly affect R_{phy} by controlling R_{rtp}.

From the diagnostic interface on commodity smartphones, we can extract the LTE physical-layer statistics including the uplink firmware buffer level (B) and uplink transport block size or TBS (P_{tbs}), for every LTE uplink subframe [23].

4.2 Adaptive Spatial Compression

POI360's adaptive compression module aim to tame the 360° video quality fluctuations caused by ROI changes. As discussed in §4.1, for ROI-based compression, the sender needs to keep updating the compression matrix $\boldsymbol{L}(t)$ based on $r_s(t) = (i_s^*, j_s^*)$, its current knowledge of the client-side ROI. Due to the end-to-end feedback latency d_f, the sender's knowledge of the viewer's ROI position $r_s(t)$ lags behind the actual value $r_c(t)$, *i.e.*, $r_s(t) = r_c(t - d_f)$. As a result, when the viewer shifts her ROI to $r_s(t)$ at time t, the sender will keep using the previous ROI $r_c(t - d_f)$ to compress the 360° video stream until $t + d_f$. In this duration, the user view has shifted to the new region, while the sender has not updated the compression level for this region yet. So the viewer observes a quality drop in ROI since time t, and then experiences a quality surge at time $t + d_f + d_v$ (where d_v denotes the one-way video frame delay), as it takes some time for the sender to be aware of an ROI change and deliver the video frames compressed with updated ROI to the viewer. Such quality fluctuations severely degrade the user experience[19].

To address this issue, POI360's adaptive spatial compression allows the sender to gauge the responsiveness of the end-to-end ROI update, and then adopt the spatial compression mode $\mathcal{F}$ that best fits current responsiveness level. Under swift ROI update, the sender can aggressively compress the non-ROI regions. On the other hand, under sluggish ROI update, the sender should create a smooth quality transition from ROI to the non-ROI regions, as the compressed content may be temporarily displayed in the new ROI. The key challenge here is to *precisely estimate the end-to-end ROI update responsiveness*. As shown in Fig. 10, this responsiveness is affected by 3 factors: *(i) The ROI feedback delay* d_f determines how long it takes for the sender to be aware of an ROI change. *(ii) The one-way video frame delay* d_v reflects how long it takes for the 360° video stream compressed with the updated ROI to arrive at the client. *(iii) The ROI stability* shows how often the user changes the ROI and by how much.

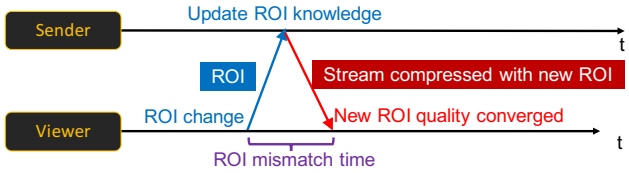


Figure 10: ROI mismatch due to network latency.

POI360 employs a single metric, the *ROI mismatch time* (M) to capture all these factors. As shown in Fig. 10, M is defined as the time interval during which the 360° video sender and client have inconsistent knowledge of the user's ROI. When the network causes higher ROI feedback latency, M will be higher. Similarly, when the network causes higher video frame delay, it takes longer for the frames with updated ROI to reach the client, leading to higher M as well. Finally, when the user switches the ROI consecutively, inconsistency becomes more severe, again leading to higher M.

POI360 measures M by monitoring the ROI video quality at the client side: Ideally, the ROI region should always enjoy the highest quality. In other words, the ROI compression level $l(i_c^*, j_c^*)$ should equal l_{min}. However, when the user quickly switches to a new ROI, her head may temporarily point to a low-quality region. Following Eq. (2), POI360 measures M at the time scale of every 360° video frame.

$$M = \begin{cases} \max(t - t_0, d_v), \text{if } l(i_c^*, j_c^*) \neq l_{min} \\ d_v, \text{otherwise} \end{cases} \quad (2)$$

where the client starts counting the time on detecting the ROI change at time t_0, and computes $M = \max(t - t_0, d_v)$ for the frame coming at time $t > t_0$ until the quality in the new ROI converges to the highest level. For the frames in which current ROI already has the lowest compression level, *e.g.*, when the user keeps the ROI, we have $M = d_v$, as the update latency for potential ROI change must be higher than current frame delay d_v. The client then uses a sliding time window to average across the frame-level measurements of M, and periodically feeds back the average M to the sender, with the feedback interval the same as the video frame interval.

After receiving M, to achieve a balance between traffic load reduction and the stability of user-perceived ROI quality, the sender then switches between K pre-defined compression modes $\mathcal{F}_1, ..., \mathcal{F}_K$ listed in the order of decreasing compression aggressiveness (smoother quality drop for regions further away from ROI). Our implementation uses $K = 8$ and empirically selects the compression mode $\mathcal{F}_{i_m}$ with $i_m = \max(8, \lceil M/200ms \rceil)$, and the constant C in Eq. (1) is selected from $[1.1, 1.2, ..., 1.8]$ for the 8 compression modes, representing 8 levels of compression aggressiveness.

4.3 Firmware Buffer Aware Congestion Control

POI360's Firmware Buffer Aware Congestion Control (FBCC) aims to address the challenge of irresponsive rate adaptation, following the solution principle in §3. In what follow, we describe the details and rationales behind our design choices.

4.3.1 Panoramic video encoding bitrate control. The core principle behind FBCC's video bitrate control is to let the sender promptly detect the overuse of cellular uplink bandwidth based on its local cellular firmware buffer occupancy, instead of waiting for the remote peer's end-to-end congestion measurement feedback. Upon detecting bandwidth overuse, the panoramic video sender immediately reduces its encoding bitrate to fit the instantaneous uplink bandwidth, which is again computed based on the PHY-layer statistics.

How to detect cellular uplink congestion? Conventional congestion control algorithms, like the GCC used in WebRTC, treat the network as a blackbox and rely on the end-to-end network delay/loss metrics to detect congestion. As a result, if a congestion happens, it takes at least one RTT for the sender of conventional rate control to detect it. In effect, due to the large buffers at cellular basestations [42], those metrics become extremely irresponsive, taking up to seconds to react to a congestion.

In contrast, FBCC is customized for interactive panoramic video services running over cellular networks, exploiting the fact that the last-hop cellular path typically bottlenecks the network path. FBCC is essentially a network-assisted rate control algorithm, which leverages the uplink firmware buffer occupancy to enable quick detection of potential congestion. Specifically, let $J \in \{0, 1\}$ be the congestion indicator with $J = 1$ showing a congestion. $B(t)$ denotes the firmware buffer occupancy at time t, and Δt denotes the report interval of firmware buffer occupancy from the phone's chipset, FBCC estimates a bandwidth overuse and potential congestion based on the following conditions:

$$J = \begin{cases} 1, \text{if } B(t-(n-1)\Delta t) > B(t-n\Delta t) \\ \quad \text{for } n = 1, \ldots, K \text{ and } B(t) > \Gamma(t) \\ 0, \text{otherwise} \end{cases} \tag{3}$$

Essentially, the FBCC estimates a congestion on detecting both of the following events: *(i)* A continuous increases of firmware buffer occupancy $B(t)$ for K consecutive reports from the chipset. To guarantee the responsiveness, POI360 uses a small $K = 10$. *(ii)* Current firmware buffer occupancy $B(t)$ exceeds a threshold $\Gamma(t)$, which is set to the long-term average buffer level and keeps being updated online.

How to throttle the rate on detecting an uplink congestion? After detecting the uplink bandwidth overuse, the rate control algorithm should reduce the video encoding bitrate to avoid potential congestion. While classical congestion-control solutions throttle the rate following empirical curves, POI360 can precisely cut the encoding bitrate to fit the instantaneous cellular uplink bandwidth. In particular, the sum of the TBS (P^w_{tbs}) for all W LTE subframes (with 1ms length) in a time window divided by the length of the time window W is the *current* PHY-layer LTE uplink throughput:

$$R_{phy} = \frac{\sum_{w=1}^{W} P^w_{tbs}}{W} \tag{4}$$

When an uplink bandwidth overuse is detected ($J = 1$), the throughput R_{phy} on this saturated uplink channel is exactly the available uplink bandwidth R_{bw}:

$$R_{bw} \begin{cases} = R_{phy}, \text{if } J = 1 \\ > R_{phy}, \text{otherwise} \end{cases} \tag{5}$$

As a result, after detecting a uplink bandwidth overuse, FBCC directly sets the encoding bitrate to T_u based on Eq. (4). Note that POI360 resorts to T_u only under congestion, because T_u represents the current rate that the client can *exploit*, but cannot indicate the potential rate it can *explore*.

Handling congestion elsewhere. In certain rare cases, congestion can happen elsewhere along the end-to-end path, *e.g.*, due to the unpredictable heavy cross traffic. POI360 detects and handles such cases by degrading to the legacy WebRTC [18] end-to-end congestion control (discussed in §7), Google Congestion Control (GCC). More specifically, let R_{gcc} denote the bitrate selected by the legacy GCC algorithm, and t^* denote the moment when a cellular uplink congestion is detected, POI360 combines both the cellular uplink congestion control and legacy end-to-end congestion control for encoding bitrate adaptation as follows:

$$R_v(t) = \begin{cases} \frac{\sum_{w=1}^{W} P^w_{tbs}}{W}, \text{if } t^* \le t \le t^* + 2RTT \\ R_{gcc}(t), \text{otherwise} \end{cases} \tag{6}$$

As shown in the above equation, *(i)* When the cellular uplink becomes the bottleneck ($J = 1$ detected at $t = t^*$), POI360 immediately reduces the encoding bitrate R_v to the cellular uplink bandwidth R_{bw} computed by Eq. (5). In this way, POI360 is more responsive than GCC as its congestion control does not wait for the reduced R_{gcc} feedback from the remote peer. Furthermore, our solution is also more accurate as it directly converges to the cellular uplink bandwidth, instead of sharply reducing the rate and probing the bandwidth again. *(ii)* When $J = 0$, the congestion happens elsewhere, or there is no congestion at all. By setting the encoding bitrate R_v to R_{gcc}, the selected bitrate from the WebRTC client, POI360 reuses the legacy WebRTC congestion control logic to handle network congestion elsewhere. Similarly, if there is no congestion, uplink congestion control will not be triggered, so we still have $R_v = R_{gcc}$.

Note that after a bandwidth overuse occurs at the cellular uplink ($t^* < t \le t^* + RTT$), legacy GCC at the client side will also detect a bandwidth overuse and then notifies the sender to throttle the rate by sharply reducing the R_{gcc} to feedback,

which arrives the sender one RTT after our algorithm reduces the rate to R_{bw}. In other words, the different responsiveness of sender-based and client-based algorithms may causes two consecutive rate reductions on a single bandwidth overuse event. To avoid this, as in Eq. (6), our design keeps forcing the video encoding bitrate to the PHY-layer uplink bandwidth R_{bw} within a duration of 2 RTTs (in cases the RTT itself changes) after an uplink bandwidth overuse is detected.

We also emphasize that FBCC reduces the 360° video encoding bitrate, rather than the transport-layer bitrate, when bandwidth overuse occurs. As shown in Fig. 9, if we throttle the transport-layer bitrate on an uplink bandwidth overuse, it simply changes the queuing location from the firmware packet buffer to the application-layer packet buffer, but the total queue length remains unchanged. The only ways to clear the queues are to *throttle the source traffic rate* and *increase the physical-layer service rate*. The encoding rate control essentially throttles the source traffic rate. In what follows, we introduce how FBCC boosts the physical-layer service rate by properly adapting the transport-layer bitrate.

4.3.2 Cellular-Link-Informed RTP Rate Control. As discussed in §3.3, conventional rate control may unnecessarily leave the cellular firmware buffer empty due to the poor interaction between the video bitrate control and transport rate control. FBCC addresses this issue by pushing the local firmware buffer level to the "sweet spot" that maximizes bandwidth utilization without causing congestion. Since it cannot directly change the firmware buffer level, FBCC indirectly controls the buffer level by controlling the buffer's input – the RTP sending bitrate. The algorithm works with a time granularity of D_p, which is the feedback interval of the buffer level from the chipset (40ms in our test device).

At the beginning of each time epoch with duration D_p, the RTP rate controller monitors the occupancy of firmware buffer $B(t)$. When it drops below the target buffer level B^*_{phy} corresponding to the optimal cellular uplink rate, it updates its current RTP sending rate $R_{rtp}(t)$ following:

$$R_{rtp}(t) = R_{rtp}(t - D_p) + \big(B^*_{phy} - B(t)\big)/D_p \qquad (7)$$

The physical meaning of Eq. (7) is to increase the RTP bitrate so that the buffer level can reach the target level when the next time epoch starts. The optimal firmware buffer level B^*_{phy} depends on the consistent uplink scheduling policy of the LTE BS, which can be learnt from previous transmissions. In the rare cases when the cellular uplink is not the network bottleneck, the video encoding bitrate $R_v(t)$ is bounded by the network bottleneck bandwidth as in Eq. (6), and we may have $R_v(t) < R_{rtp}(t)$, *i.e.*, the uplink firmware buffer can still drain out, but that prevents the congestion at the network bottleneck. We also note that this design does not affect the fairness to other cellular users, because the LTE BS controls the fairness in uplink scheduling.

5 IMPLEMENTATION

Browser-based adaptive spatial compression. We implement POI360's spatial compression algorithm using HTML5. Specifically, every raw 360° video frame is divided to small tiles (12×8 tiles in our implementation) by writing each tile to a separate HTML5 canvas. The size of each tile is compressed based on current compression mode $\mathcal{F}$. The compressed tiles are then stitched together in one HTML5 canvas, in which the sender also embeds its current compression mode and its knowledge of the viewer's ROI. Finally, the browser's WebRTC module captures and encodes the outgoing video stream from this canvas. After receiving the compressed video stream, the client first infers the sender's compression matrix using the information embedded inside each received video frame and then unfolds this compressed frame accordingly. Then the client uses WebGL to render the unfolded video frame as stereoscopic view for left and right eyes, which can be viewed with stereoscope VR headsets like Google cardboard. Since the 360° video stream is captured from real world, rather than generated based on a real-time 3D engine, and the ROI-compressed video frame dimension become comparable to that of conventional video, the computational overhead and latency of 360° video encoding and rendering for POI360 are comparable to WebRTC-based conventional video telephony like Google Hangout.

Besides rendering the stream, the client also keeps updating current ROI based on the user input like the motion sensor reading on the phone, and periodically feeds back current user ROI to the sender via WebRTC data channel. The feedback interval is set to the video frame interval [1]. Meanwhile, the client also keeps monitoring the responsiveness (M discussed in §4.2) of the end-to-end ROI update, which is also fed back via the WebRTC data channel. Based on M, the sender then adapts its compression mode following §4.2.

Firmware Buffer Aware Congestion Control. POI360's cellular-link-informed rate control obtains the physical-layer statistics from the diagnostic interface, which is available on most commodity phones. So it requires no hardware modifications and is readily deployable. Following the implementation of MobileInsight[23], the POI360 prototype uses our customized real-time log decoder to read the phone's diagnostic interface and obtains the LTE uplink TBS and the uplink firmware buffer level for every 40ms. These cellular physical layer statics are then written to shared memory which is also accessed by POI360's FBCC algorithm. We integrate FBCC into the WebRTC module in the open-source Chromium browser by modifying the GCC implementation in WebRTC source code.

[1]Since both the WebRTC data channel and video stream run over UDP, current LTE BS gives the same priority to the ROI feedback and video data. In future 5G networks with more flexible flow control, the BS may schedule the ROI feedback at higher priority to further improve performance.

360° video telephony performance measurement. *(i) End-to-end video frame delay measurement.* The core idea is to embed the sending timestamp of a video frame inside the frame itself, the receiver then extracts the sending time from the received frame and subtracts it from local time to compute the delay. The sender and receiver are synchronized using NTP. Each decimal bit of the sending timestamp (with millisecond resolution) is encoded using a colored square block, with the number from 0 to 9 mapping to 10 colors with uniform separation in the RGB code space. These blocks are then appended to the side of the outgoing video frame.At the client side, the RGB values of pixels within each colored block region of the received frame are averaged and then mapped back to the decimal number.

(ii) 360° video quality measurement. Video quality measurement requires comparison between the original and the received video frame. However, for 360° video, the users only care about the quality within ROI, as other regions are not rendered. Therefore our measurement system matches sender-side and client-side frames in both time and space domain to facilitate 360° video quality measurement. In particular, the client only dumps the ROI of its received frame, along with the corresponding ROI position (i_c^*, j_c^*). The sender, on the other hand, may have stale ROI knowledge due to the network latency, so it has to dump the entire frame. Finally, in the ROI comparison, we crop the ROI region from the original frame based on the client's corresponding ROI position.

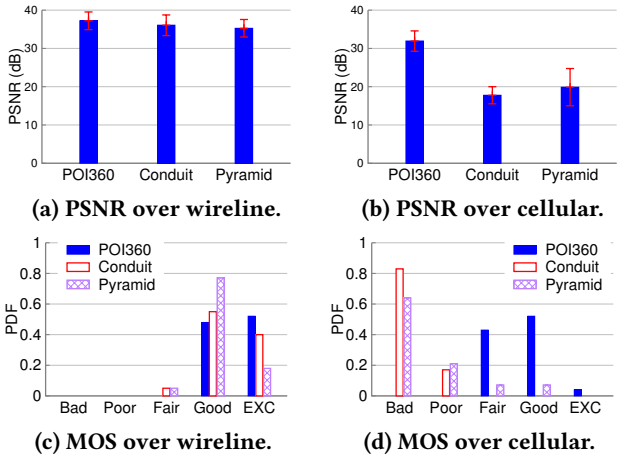
6 POI360 EVALUATION

In this section, we first use micro-benchmark experiments (§6.1) to validate POI360's design components, including the adaptive spatial compression (§6.1.1) and the FBCC rate control (§6.1.2), and demonstrate the individual performance gain contributed by each of them, Then we conduct extensive system level tests for our real-time POI360 prototype under various network conditions (§6.2). Our experiments run on a commercial LTE network using LG Nexus 5 D820 smartphones. To obtain representative human-controlled ROI, we invite 5 users to participate in the experiments. Since the behavior of an user's ROI is highly correlated with the video content, we use different 360° video for each user to avoid overfitting. To guarantee repeatable 360° video traffic load in experiments, we use virtual webcam[41] to live stream the same 360° video with 4K resolution when repeating the experiments for a user. In our experiments, each 360° video telephony session runs for 5 minutes, and each user repeats the test session for 10 times. Due to the limited LTE bandwidth, other network applications running on the same device can degrade POI360 performance. To isolate such effects, we turn off other data consuming applications on the phone.

6.1 Microbenchmarks

MOS Video Quality	PSNR Range (dB)
Excellent	> 37
Good	$31 \sim 37$
Fair	$25 \sim 31$
Poor	$20 \sim 25$
Bad	< 20

Table 1: Mapping PSNR to Mean Opinion Score (MOS).



(a) PSNR over wireline. **(b) PSNR over cellular.**

(c) MOS over wireline. **(d) MOS over cellular.**

Figure 11: Comparing video quality under POI360 compression and benchmark algorithms.

6.1.1 Adaptive Spatial Compression. We first verify the adaptive spatial compression. To highlight the effectiveness of POI360's design customized for cellular networks, the same set of experiments are also repeated over a wireline network on a university campus. Specifically, we run a POI360 session with two laptops, and the Internet access of both laptops are provided by either wireline connection or LTE. To isolate the impact of network protocols, we use GCC, WebRTC's default rate control (§2), as the transport layer.

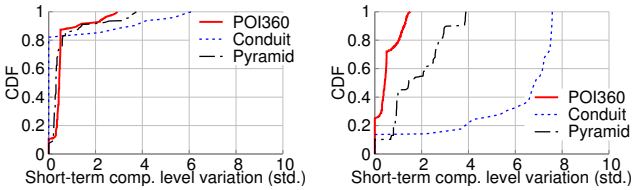
We compare POI360 with two benchmark panoramic video compression schemes including Conduit[1] and Pyramid encoding[7]. Conduit crops the ROI region from the panorama video frame and only streams the cropped parts. To avoid displaying blank frame, we still send the non-ROI regions for Conduit but with the lowest possible quality. Pyramid encoding centers the panorama video frame at the ROI, and then allocates the center region the highest quality and aggressively compresses the contents in the 4 corners based on their distance to the center. Based on our system mode (§4.1), Conduit can be considered as a very aggressive compression mode with a sharp quality distribution curve, whereas Pyramid encoding adopts a conservative compression scheme with smooth quality distribution, so it can better handle the ROI shifting at the cost of higher traffic load. However, both algorithms are incapable of dynamically adapting the compression modes based on the network conditions. Besides, motion-prediction based compression[30] does not work in our application scenario (§8), thus is not compared here.

User-perceived video quality in ROI. We first evaluate the user-perceived ROI quality based on the aforementioned setup. First, Fig. 11a and 11b plot the average PSNR within the ROI region over both cellular and wireline networks, with error bars showing the standard deviation of PSNR value. We can observe that POI360's compression algorithm achieves the highest PSNR within the ROI region. In wireline networks, all the algorithms perform reasonably well. However, Conduit and Pyramid suffer from significant quality loss when running over cellular networks, with 11 to 13 dB lower PSNR than POI360. Such disparate performance comes from POI360's capability to adapt the compression modes based on the network latency that affects the end-to-end ROI update responsiveness. POI360 can effectively switch to the proper spatial compression mode that best fits the current network condition. In contrast, when a viewer changes ROI in Conduit, the remote peer cannot update its knowledge of the viewer's ROI in time. So the viewer can only see compressed regions with low quality. Pyramid shows slightly higher PSNR than Conduit, as it has a smoother spatial quality distribution, and hence less quality drop during the ROI shift.

To understand the viewer's subjective experience, in Fig. 11c and 11d, we further plot the PDF of the Mean Opinion Score (MOS) computed based on the frame-level PSNR, following the well-known relation between the MOS and PSNR[36] summarized in Table 1. We see that POI360 achieves the best MOS over both cellular and wireline networks. In particular, over cellular networks, 52% of its frames have good quality while 4% have excellent quality. Conduit does not provide any frame with good or excellent quality, and Pyramid only has 7% frames with good quality. Even in wireline networks, POI360 has more than half of the frames with excellent quality and the rest of the frames with good quality, substantially outperforming both Conduit and Pyramid.

We emphasize that POI360's performance gain mainly comes from its principle of *adaptive* compression. The compression itself is taking effect just like other schemes. For example, given a 360° video with 12.65Mbps raw bitrate, the received stream bitrate at the POI360 client is only around 3Mbps, reduced by 76% (more in §6.1.2 and Fig. 16a).

Video quality stability in ROI. Besides video quality itself, the short-term stability of video quality is also vital for user experience[19]. We characterize this *stability* by the standard deviation of the ROI compression level in a 2 second sliding window. The CDF plot in Fig. 12a shows that all the algorithms have small quality variation in wireline networks with relatively stable network condition and small RTTs. In contrast, in cellular networks (Fig. 12b), Conduit and Pyramid show poor stability, with 5× and 14× higher *std.* than POI360. This is because the large and unstable RTT of cellular networks makes the end-to-end ROI update laggy,



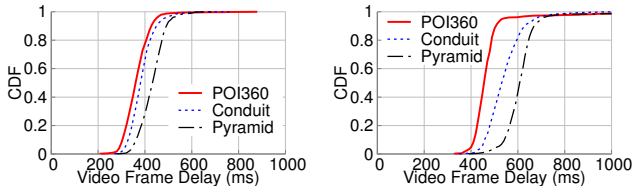
(a) ROI region compression level variation over wireline. **(b) ROI region compression level variation over cellular.**

Figure 12: Short-term ROI compression level variation (std. value of ROI compression level in a 2s window).

causing the fluctuating compression level in the actual ROI. The problem is especially severe for Conduit, as it only has 2 compression levels, thus ROI shifting triggers unacceptable video quality oscillation between the high/low levels. Using more compression levels, Pyramid smooths out the transition, thus achieving better stability than Conduit. However, its rigid spatial compression mode still cannot fit all conditions in the highly dynamic cellular networks. In contrast, POI360 achieves the best stability as it can always adapt to the compression mode best fitting current network condition.

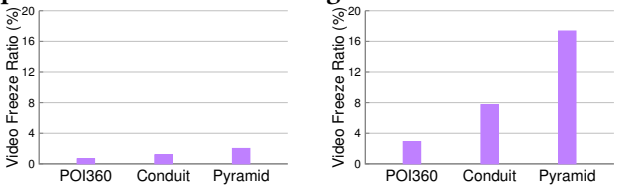
360° video frame delay. We now analyze the frame-level delay performance collected from the same set of experiments. Fig. 13a plots the CDF of frame delay over the wireline network. We can observe that POI360's compression algorithm achieves the lowest video delay. Furthermore, in cellular networks (Fig. 13b), POI360 achieves a median delay of 460ms, 15% less than Conduit which aggressive compression, because its adaptive design can switch to more aggresive compression modes than Conduit under bad network condition, or prioritize the video quality when the network allows. In other words, POI360's better ROI quality and stability come without sacrificing the crucial delay performance. This delay is also comparable to the conventional video telephony delay over cellular networks measured in existing works[46]. We emphasize that end-to-end frame delay is not frame interval, as shown in Fig. 2, *e.g.*, a video stream can be delayed by 460*ms* while keeping a 36FPS frame rate. Pyramid encoding has higher video delay than Conduit, as it compresses the non-ROI regions less aggressively. In contrast, POI360 strikes a balance between video quality and delay: When the end-to-end ROI update is responsive, it can aggressively crop and stream only the high-quality ROI like Conduit; When the ROI update becomes sluggish, it can distribute the video quality more evenly across the frame like Pyramid encoding.

Video frame freezing ratio. Based on the video frame delay, we then compute the frame freezing ratio, the most crucial user experience metric. We define *freezing ratio* (F_R) as the percentage of video frames that experience higher than 600ms delay. Fig. 14a shows that all algorithms work properly in wireline networks, with F_R below 2%, and POI360 has the lowest F_R of 0.6%. However, in cellular networks



(a) Frame delay (wireline). (b) Frame delay (cellular).

Figure 13: 360° video frame delay under POI360 compression and benchmark algorithms.



(a) Freeze ratio (wireline). (b) Freeze ratio (cellular).

Figure 14: 360° video freeze ratio under POI360 compression and benchmark algorithms.

(Fig. 14b), both Conduit and Pyramid encoding fail with an unacceptable F_R of 8% to 17%. In contrast, POI360's adaptive compression tames the F_R below 3% under the network dynamics.

In sum, *POI360's adaptive compression substantially outperforms the benchmark algorithms in terms of video quality, stability and delay, especially in cellular networks with long RTT and highly dynamic network conditions.* However, there is still a performance gap between cellular and wireline , especially in terms of freezing ratio. In what follows, we show how POI360's cross-layer rate control helps bridge this gap.

6.1.2 Firmware Buffer Aware Congestion Control. Recall that POI360's FBCC module aims to improve the cellular uplink bandwidth utilization by pushing the firmware buffer occupancy to a proper level. To verify its effectiveness, we repeat the panoramic video telephony sessions with each lasting for 200 seconds for both POI360 over FBCC and POI360 over GCC—the default WebRTC rate control used in the foregoing experiments. Both configurations share the same application-layer dynamic panoramic video compression module in §6.1.1. The scatter plot in Fig. 15 shows the firmware buffer level and the per-second uplink TBS (throughput) collected across the panoramic video telephony sessions. We split this figure into 3 regions: *(i) the low usage region* with uplink throughput below 2 Mbps; *(ii) the high usage region* where the uplink throughput exceeds 2Mbps and keeps increasing with the buffer level; *(iii) the overuse region* where the uplink throughput no longer increases with the firmware buffer level as the uplink is saturated. It is clear that FBCC maintains the buffer level around the "sweet spot" in the high usage region that is far from congestion but high enough to secure sufficient bandwidth utilization. In contrast, GCC's buffer occupancy stays at the low usage region

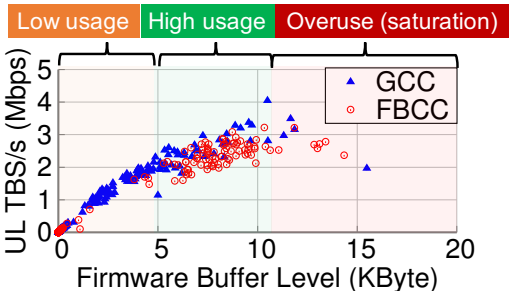
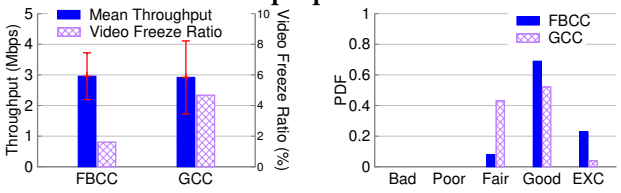


Figure 15: FBCC is allocated higher uplink TBS than GCC as it maintains a proper firmware buffer level.



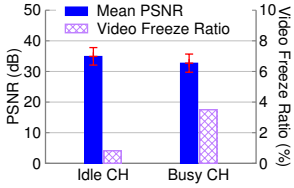
(a) Video freeze ratio. (b) Video quality (MOS).

Figure 16: Comparing POI360 performance with FBCC or GCC (WebRTC's default rate control).

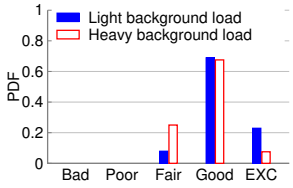
for a substantial fraction of the samples, indicating that it severely underutilizes the network bandwidth.

We proceed to evaluate the impact of the transport layer rate control on the performance of panoramic video telephony. Following the same experimental setup discussed at the beginning of this microbenchmark, we compare POI360 running over FBCC against POI360 running over legacy GCC. From Fig. 16a, we can observe that the throughput of both configurations are almost identical at around 3Mbps, as they share the same compression algorithm which compresses the panoramic video stream bitrate to such a level below the current cellular uplink bandwidth.

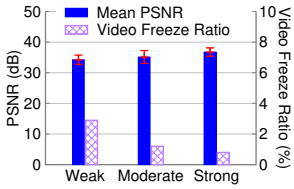
However, the throughput variation differs a lot. Specifically, GCC's throughput has a 1.2Mbps standard deviation, 57% higher than FBCC. This unstable throughput is mainly caused by its inefficient adaptation logic, which explores the network bandwidth by gradually increasing the sending rate, and reacts to the bandwidth overuse by sharply throttling the traffic bitrate. In contrast, FBCC converges to the network bandwidth quickly as it can directly estimate the cellular uplink bandwidth based on the firmware buffer level and TBS following Eq. (5) when the buffer status indicates a congestion, and then responsively throttles the rate to the estimated bandwidth. As a result, the extra cross-layer information helps FBCC achieve a low F_R of around 1.6% over cellular networks, which is significantly lower than the F_R of GCC (4.7%), and is even close to the wireline case (Fig. 14b). The stability of throughput also translates into high video quality. Fig. 16b shows the MOS quality computed based on the PSNR collected from the same experiment. We see that FBCC has much higher percentage of video frames with good (69%) and excellent (23%) quality than GCC. In contrast,



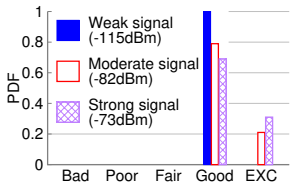
(a) Freeze ratio & PSNR with different background load.



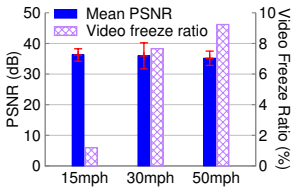
(b) Video quality (MOS) with different background load.



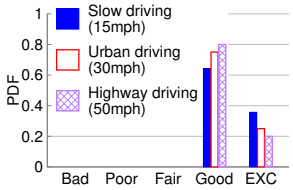
(c) Freeze ratio & PSNR with different signal strength.



(d) Video quality (MOS) with different signal strength.



(e) Freeze ratio & PSNR at different driving speed.



(f) Video quality (MOS) at different driving speed.

Figure 17: System-level POI360 evaluation.

GCC has more than 40% of the frames with only fair quality, as its throughput frequently vibrates to low levels.

6.2 System Level Evaluation

We now integrate POI360's design components and conduct a system level evaluation under various network conditions that lead to different level of network bandwidth and ROI feedback latency.

Different background traffic load. Since POI360 runs on cellular networks with the uplink being the typical bottleneck, we verify its resilience under different uplink traffic load within the same cell. We conduct the experiments in two campus environments: *(i) light traffic load environment* during early morning when most users are off campus; *(ii) heavy traffic load environment* in the noon just after class, when most roam around campus with their cell phones. All experiments are performed at the same static location to rule out the influence of signal strength and mobility.

Fig. 17a plots the video freezing ratio F_R and PSNR. We observe that the overall F_R is only around 1% under light traffic load. Even during the busiest hour, the freezing rate is only around 4%, and PSNR drops by 2 dB. Under light traffic load, the MOS metric shows a relatively larger fraction of frames with excellent quality. But even with heavy competing traffic, majority of the frames are in either excellent or good condition, and none is poor or bad condition. Therefore, our results demonstrate that POI360 is robust to the background traffic load. Such robustness owes to its awareness of cellular link statistics like the uplink firmware buffer level, and its ability to effectively maintain an adequate uplink rate by manipulating the buffer level.

Different LTE channel quality. Since POI360 aims to realize the ubiquitous capturing and streaming of 360° video, it is important to understand the impact of the location-dependent cellular link quality on its performance. To create different channel qualities, we place our prototype at different locations and evaluate the performance under different levels of received signal strength (RSS): *(i) weak signal* inside a concrete parking garage, where the RSS is −115dB as reported by the phone's firmware. *(ii) moderate signal* at an outdoor parking lot partially shadowed by a tall building (−82dB RSS). *(iii) Strong signal* in an open parking lot without tall buildings nearby (−73*dB* RSS). The experiments are conducted during weekends when the cellular channel is mostly idle.

Fig. 17c shows the resulting PSNR and video freeze performance. We first see that the F_R is not affected much by the RSS. Even under weak channel, the F_R is below 3%. However, this comes at the cost of lower video quality (Fig. 17d). Under a weak channel, none of the frames show excellent quality, whereas the strong channel has 31% of the frames with excellent quality. The different trends for video freeze and video quality can be explained by the stability of RSS. Although the channel can be weak which forces POI360 to use low video quality, as long as the RSS does not fluctuate, POI360's rate control can always converge to the network bandwidth and stably stream the video without freezing.

Different mobility level. One potential application of POI360 is to enable 3D experience sharing through drones or inside moving vehicles. Different mobility levels in turn affects the LTE network condition. To evaluate the impact, we place our experimental platform inside a moving vehicle with 3 different speed configurations: *(i)* 15 mph, a typical slow driving speed within the residential area; *(ii)* 30 mph, driving along a road in an urban area; *(iii)* 50 mph, driving along a highway. We repeat each test 10 times along the same route. During the experiments, the phone is under seamless LTE coverage.

Fig. 17e shows the performance in terms of F_R. We see that slow driving has almost no impact on F_R compared with the static experiments. But F_R increases to 7% under city driving speed and 9% on the highway. Fig. 17f further plots the MOS video quality computed based on the PSNR, we can observe that although the video quality decreases under higher mobility level, even under the speed of 50 mph, all frames still have either excellent (20%) or good (80%) quality, which mainly owes to the less building blockage and thus higher RSS (typically −60dB) around the highway test route.

7 RELATED WORK

The past decade of research in 360° video delivery focused on the encoding aspect [4], *i.e.*, mapping the original sphere view into various planar formats [8, 10, 11, 26, 31]. The concept of splitting an 360° frame into tiles and only delivering a fraction of the tiles based on the viewer's ROI has been investigated [33] to ensure high ROI resolution under certain bandwidth and latency constraint. To date, 360° video streaming mainly deliver pre-captured videos on demand, which can tolerate second-level buffering latency [20]. Through simulation, Qian *et al.* [30] verified the feasibility of delivering on-demand 360° videos through cellular networks.

Driven by portable 360° cameras, real-time immersive experience sharing is emerging as a new class of applications. Through a 360° camera mounted on a drone [22, 29], headset [25], or handheld device [34], real-time 360° scenes can be teleported to remote users, who can experience first-person 360° view with a VR HMD. Microsoft's Holoportation [27] represents the state-of-the-art in streaming 360° videos for telepresence. However, current Holoportation requires 30-50 Mbps bandwidth, and only works within Wi-Fi range.

Ultimately, 360° videos call for a more agile adaptation to the network conditions than conventional video as its heavy traffic is more sensitive to the bandwidth variations. Proprietary video chat systems commonly build their own rate control atop of UDP [46], and measurement studies reveal severe performance limitations, such as sluggish response to bandwidth change [5] and frequent freezing [46] especially over cellular networks. Recently, the IETF chartered the RMCAT working group [17] to standardize congestion control for real-time media. Google Congestion Control (GCC) [12] has been a leading proposal in RMCAT, and acts as the media transportation framework in mainstream browsers like Chrome and Firefox. Alternative proposals [47, 48] regulate sending rate based on explicit congestion notification (ECN), but require ECN support on intermediate routers. In recent Linux kernel, Byte Queue Limits (BQL) can avoid starvation and unnecessary queuing for the driver queue [37]. But it has no access to the cellular firmware buffer level or the link quality. In [44], Xu *et al.* proposed a machine learning model to predict cellular network performance based on historical throughput/latency, which, however, still needs at least one RTT to detect a network change as an end-to-end solution.

Parallel to the interactive media research, on-demand video streaming protocols are converging towards the HTTP based dynamic adaptive streaming (DASH) [38]. The key issue is to accurately estimate the available bandwidth (by monitoring playback buffer level [16] or historical throughput [19], *etc.*), and then choose a video rate below it. Recently, piStream [43] proposed to adapt the video bitrate according to LTE downlink resource utilization. Such on-demand video streaming systems leverage the large client buffer of several seconds to absorb bandwidth estimation inaccuracy. In contrast, interactive video has orders of magnitude tighter delay constraints , thus calls for a responsive rate control design.

8 DISCUSSION & FUTURE WORK

ROI prediction. *(i)* For rendering local 360° video, motion-based ROI prediction [2, 21] helps smooth the display quality. However, such prediction only works reliably at a short time scale. In typical use, the average head angular velocity is 60 deg/sec while acceleration can go up to 500 deg/sec^2 as reported by Oculus [21], *i.e.*, the head position after 120 ms is unpredictable, which is below the typical video latency over LTE. *(ii)* For on-demand 360° video multicast, recent work [3] suggests using the motion patterns learnt from previous users to predict the ROI of current user, as users perform similar motion patterns given the same video content. However, 360° video telephony is different from on-demand streaming, as the content is generated in real time. Therefore there are no existing contents to learn from.

Improving the ROI update responsiveness. Although POI360 smooths out the ROI quality transition by adaptive compression, it still takes the cellular RTT before the quality of new ROI converges. The key to reduce this convergence time is shortening the end-to-end network path. In current 4G LTE network, traffic still goes to the Internet even both ends of the video telephony connect to the same BS. In future works, mobile edge computing can be used to enable the relaying at the edge BS, thus significantly shortens the path and accelerate the quality convergence of POI360.

9 CONCLUSION

Although on-demand 360° video streaming for recorded contents has been studied extensively, real-time interactive 360° video telephony remains mostly unexplored. In this paper, we investigate the feasibility of realizing interactive 360° video telephony services over LTE cellular networks, and present our readily-deployable solution POI360, which employs adaptive compression to balance the traffic load reduction and user-perceived video quality. POI360 also features a novel cellular-link-informed rate control algorithm to boost the responsiveness to potential uplink congestion. Finally, a real-time prototype of POI360 is implemented, which validates our design in various network conditions. We believe this work can inspire new applications that bring the interactive video services to the 360° world.

ACKNOWLEDGMENTS

We appreciate the anonymous reviewers for their insightful comments. This research was supported in part by a Google Faculty Research Award, and by the NSF under Grant CNS-1506657, CNS-1518728, CNS-1343363 and CNS-1350039.

REFERENCES

[1] Aakash Patel and Gregory Rose. 2015. Conduit: Efficient Video Compression for Live VR Streaming. (2015). https://avp.github.io/conduit/

[2] Ronald Tadao Azuma. 1995. Predictive Tracking for Augmented Reality. In *Ph.D. dissertation, University of North Carolina at Chapel Hill*.

[3] Yanan Bao, Tianxiao Zhang, Amit Pande, Huasen Wu, and Xin Liu. 2017. Motion-Prediction-Based Multicast for 360-Degree Video Transmissions. In *IEEE SECON*.

[4] J. Chakareski. 2013. Adaptive Multiview Video Streaming: Challenges and Opportunities. *IEEE Communications Magazine* 51, 5 (2013).

[5] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. 2011. Skype Video Congestion Control: An Experimental Investigation. *Computer Networks* 55, 3 (2011).

[6] Eduardo Cuervo and David Chu. 2016. Poster: Mobile Virtual Reality for Head-mounted Displays With Interactive Streaming Video and Likelihood-based Foveation. In *Proceedings of ACM MobiSys*.

[7] Evgeny Kuzyakov and David Pio. 2016. Next-generation video encoding techniques for 360 video and VR. (2016). https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding

[8] Facebook. 2016. Cubemap Transform Open Source Code. (2016). https://github.com/facebook/transform

[9] Facebook. 2016. Facebook 360 Video. (2016). https://facebook360.fb.com

[10] Facebook. 2016. Next-generation video encoding techniques for 360 video and VR. (2016). https://code.facebook.com/posts/1126354007399553/next-generation-video-encodin

[11] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. 2016. Tiling in Interactive Panoramic Video: Approaches and Evaluation. *IEEE Transactions on Multimedia* 18, 9 (2016).

[12] Google. 2015. A Google Congestion Control Algorithm for Real-Time Communication. https://tools.ietf.org/html/draft-alvestrand-rmcat-congestion-03. (2015).

[13] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding On-device Bufferbloat for Cellular Upload. In *Proceedings of the 2016 ACM on Internet Measurement Conference*.

[14] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. 2013. Handling Packet Loss in WebRTC. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE.

[15] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proc. of ACM SIGCOMM*.

[16] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. of ACM SIGCOMM*.

[17] IETF. 2015. RTP Media Congestion Avoidance Techniques (RMCAT). https://datatracker.ietf.org/wg/rmcat/documents/. (2015).

[18] IETF/W3C. 2015. WebRTC. (2015). https://webrtc.org/

[19] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proc. of ACM CoNEXT*.

[20] Ngo Quang Minh Khiem, Guntur Ravindra, and Wei Tsang Ooi. 2011. Towards Understanding User Tolerance to Network Latency in Zoomable Video Streaming. In *Proceedings of the ACM International Conference on Multimedia*.

[21] Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. 2014. Head Tracking for The Oculus Rift. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*.

[22] Nick Lavars. 2016. Exo360 Drone Shoots VR Content on the Fly. (2016). http://newatlas.com/exodrone360-vr-content/43854/

[23] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *ACM MobiCom*.

[24] M. Makar, A. Mavlankar, P. Agrawal, and B. Girod. 2010. Real-Time Video Streaming With Interactive Region-of-Interest. In *IEEE International Conference on Image Processing (ICIP)*.

[25] Mariella Moon. 2016. 360fly Puts 4K Video Cams on Helmets. (2016). https://techcrunch.com/2016/10/06/oculus-shows-off-facebook-messenger-video-calls-in-virtual-reality/

[26] Khiem Quang Minh Ngo, Ravindra Guntur, and Wei Tsang Ooi. 2011. Adaptive Encoding of Zoomable Video Streams Based on User Access Pattern. In *Proceedings of the Annual ACM Conference on Multimedia Systems (MMSys)*.

[27] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien 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 ACM Symposium on User Interface Software and Technology (UIST)*.

[28] Derek Pang, Sherif Halawa, Ngai-Man Cheung, and Bernd Girod. 2011. Mobile Interactive Region-of-interest Video Streaming with Crowd-driven Prefetching. In *Proceedings of International ACM Workshop on Interactive Multimedia on Mobile and Portable Devices*.

[29] Ben Popper. 2016. Drones and Virtual Reality Headsets are a Delicious Combination. (2016). http://www.theverge.com/2014/11/24/7274997/parrot-bebop-drone-virtual-reality

[30] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 Video Delivery over Cellular Networks. In *Proceedings of the Workshop on All Things Cellular*.

[31] Ngo Quang Minh Khiem, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi. 2010. Supporting Zoomable Video Streams with Dynamic Region-of-interest Cropping. In *Proceedings of the ACM SIGMM Conference on Multimedia Systems (MMSys)*.

[32] Richoh. 2016. Theta S 360 Mobile Camera. (2016). https://theta360.com/en/about/theta/s.html

[33] Patrice Rondao Alface, Jean-François Macq, and Nico Verzijp. 2012. Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach. *Bell Labs Technical Journal* 16, 4 (2012).

[34] Adam Ryder. 2016. VR Action Camera Field Test: Ricoh Theta S and 360Fly 4K. (2016). http://www.popphoto.com/hands-on-with-two-360deg-action-cameras

[35] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory Zelinsky. 2016. Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices. In *Proceedings of the International Conference on Multimedia Systems (MMSys)*.

[36] Sayandeep Sen, Syed Gilani, Shreesha Srinath, Stephen Schmitt, and Suman Banerjee. 2010. Design and Implementation of an "Approximate" Communication System for Wireless Media Applications. In *SIGCOMM*.

[37] Dan Siemon. 2013. Queueing in the Linux Network Stack. *Linux Journal* (2013).

[38] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. In *IEEE Multimedia*.

[39] Xinding Sun, J. Foote, D. Kimber, and B. S. Manjunath. 2005. Region of Interest Extraction and Virtual Camera Control Based on Panoramic Video Capturing. *IEEE Transactions on Multimedia* 7, 5 (2005).

[40] John Tan. 2015. WebRTC Mobile: Facebook and Slack Are In. (2015). https://www.sinch.com/opinion/webrtc-facebook-google-firefox/

[41] Umlaeute. 2017. v4l2loopback - A Kernel Module to Create V4L2 Loopback Devices. (2017). https://github.com/umlaeute/v4l2loopback

[42] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *USENIX NSDI*.

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

[44] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proc. of ACM MobiSys*.

[45] YouTube. 2016. Uploading 360-degree Videos. (2016). https://support.google.com/youtube/answer/6178631?hl=en

[46] Chenguang Yu, Yang Xu, Bo Liu, and Yong Liu. 2014. "Can you SEE me now?": A Measurement Study of Mobile Video Calls. In *Proc. of IEEE INFOCOM*.

[47] Jing Zhu, R. Vannithamby, C. Rodbro, Mingyu Chen, and S. Vang Andersen. 2012. Improving QoE for Skype video call in Mobile Broadband Network. In *Proc. of IEEE Global Communications Conference (GLOBECOM)*.

[48] Xiaoqing Zhu and Rong Pan. 2013. NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video. In *International Packet Video Workshop (PV)*.